Appendix C

Title:
Flexible Peripherals Interconnect Bus
Version 3.2

1.1 FPI-Bus Features

The FPI-Bus is designed with requirements of high-performance systems in mind. The features are:

- Core independent
- Multi-master capability (up to 16 masters)
- Demultiplexed operation
- Clock synchronous
- Peak transfer rate of up to 800 MBytes/s (@ 100 MHz bus clock)
- Address and data bus scalable (address bus up to 32 bits, data bus up to 64 bit)
- 8-/16-/32- and 64 bit data transfers
- Broad range of transfer types from single to multiple data transfers
- Split transaction support for agents with long response time
- Burst transfer capability
- EMI and power consumption minimized

FPI-Bus does **not** provide:

- Cache coherency support
- Broadcasts
- Dynamic bus sizing
- Unaligned data accesses

1.2 FPI-Bus Overview

FPI-Bus is an on-chip bus to be used in modular, highly integrated microprocessors and microcontrollers (*systems-on-chips*). FPI-Bus is designed for memory mapped data transfers between its bus agents. Bus agents are on-chip function blocks (modules), equipped with a FPI-Bus interface and connected via FPI-Bus signals. A FPI-Bus agent acts as a FPI-Bus master when it initiates data read or data write operations once bus ownership has been granted to the agent. A FPI-Bus agent which is addressed at a FPI-Bus operation acts as a FPI-Bus slave when it performs the requested data read or write operation. Typical masters are processor modules, coprocessors, DMA controller or the external bus interface. Typical slaves are on-chip peripherals (**Figure 1-1**).

To increase the availability of the bus, read transaction may be split into two independent transfers: first the transfer request with some additional information on data type and block size and later the transfer of the requested data from the slave to the master. For the second part the slave will become master on the bus and the previous master the slave. Agents which support this type of transfers are called master-slave agents (refer to 3.4.5"Master-Slave Agents,").
Memory modules may be implemented as pure slaves (normal case) or as master slave modules, depending on their speed.
Other examples for FPI-Bus agents that have to provide master as well as a slave functionality are

coprocessors. A coprocessor may need to be initialized by the processor before it can be started. To initialize, the processor has to write values into the coprocessor's registers via the FPI-Bus interface. In this case the FPI-Bus interface of the coprocessor operates as a slave. When the coprocessor is allowed to run, it reads from or writes to memory or communication interfaces by its own. The FPI-Bus interface now acts as a master.

To operate, FPI-Bus requires an additional bus controller which ensures the bus protocol. It could do e.g. time-out and slave access control or protocol handling for all simple slave devices.
The bus controller functionality will be implementation dependent due to more or less centralized control approach.

The FPI-Bus protocol is oriented towards fast FPI-Bus agent accesses as well as to a high transfer bandwidth. A low-overhead protocol guarantees short response times at FPI-Bus accesses which are needed for time-critical applications. Multiple data transfers allow FPI-Bus to operate at a high bandwidth.

There are three types of agents possible on the FPI-Bus (see **Figure 1-1**):

- Master agents which can initiate and control transactions.
- Slave Agents, which only support simple read and write of registers and are not dealing with the bus protocol. This has to be done e.g. by a bus controller and
- Master-Slave Agents, which will support advanced features like split read transfer support and error handling. Depending on the type of transaction these agents may act as master or slave or both.

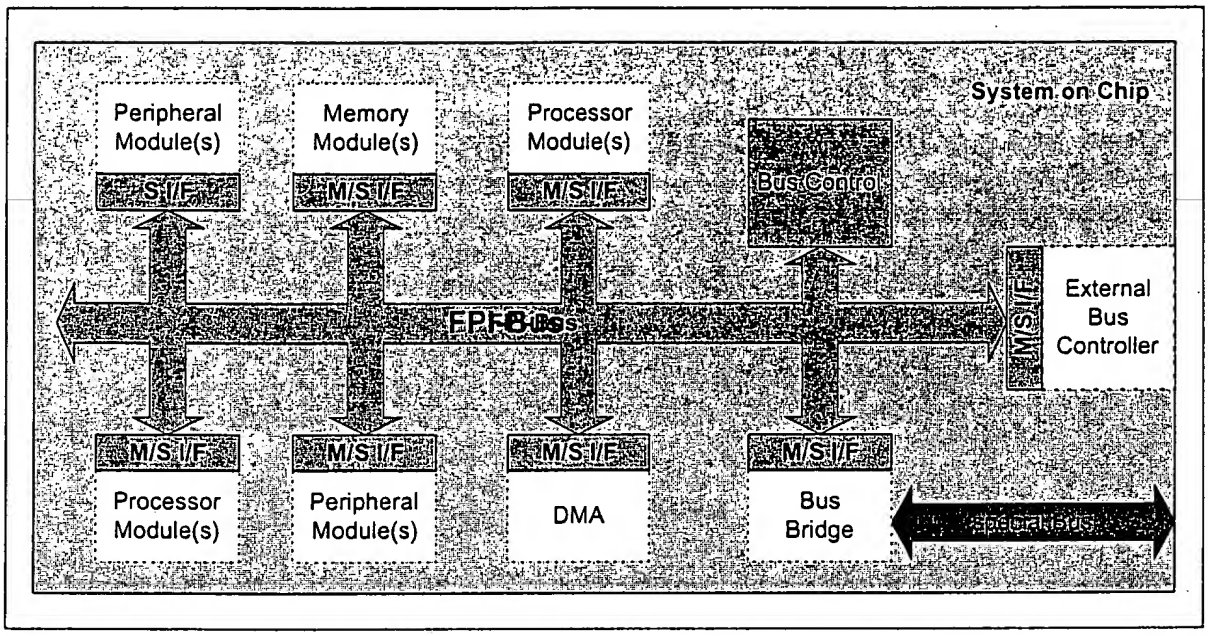

Figure 1-1: Examples for Modules of a FPI-Bus system

The FPI-Bus is designed to be flexible enough to support different processor cores, provide high data bandwidth and high availability. It can interconnect all kinds of internal and external peripherals and memories to different active bus agents like CPUs, DMA/PEC controllers and coprocessors. Special care was taken to reduce power consumption and EMI.

This is achieved by providing:

- scalability of address and data bus
- flexible bus protocol, only subsets may be implemented or optional extensions defined can be incorporated
- support for split read transfers (request of transfer and transfer of the data may be separated)
- bus signal hold circuitries for most bus signals (to prevent undefined state on undriven bus lines)

Figure 1-2 shows an example for a simple FPI-Bus implementation.

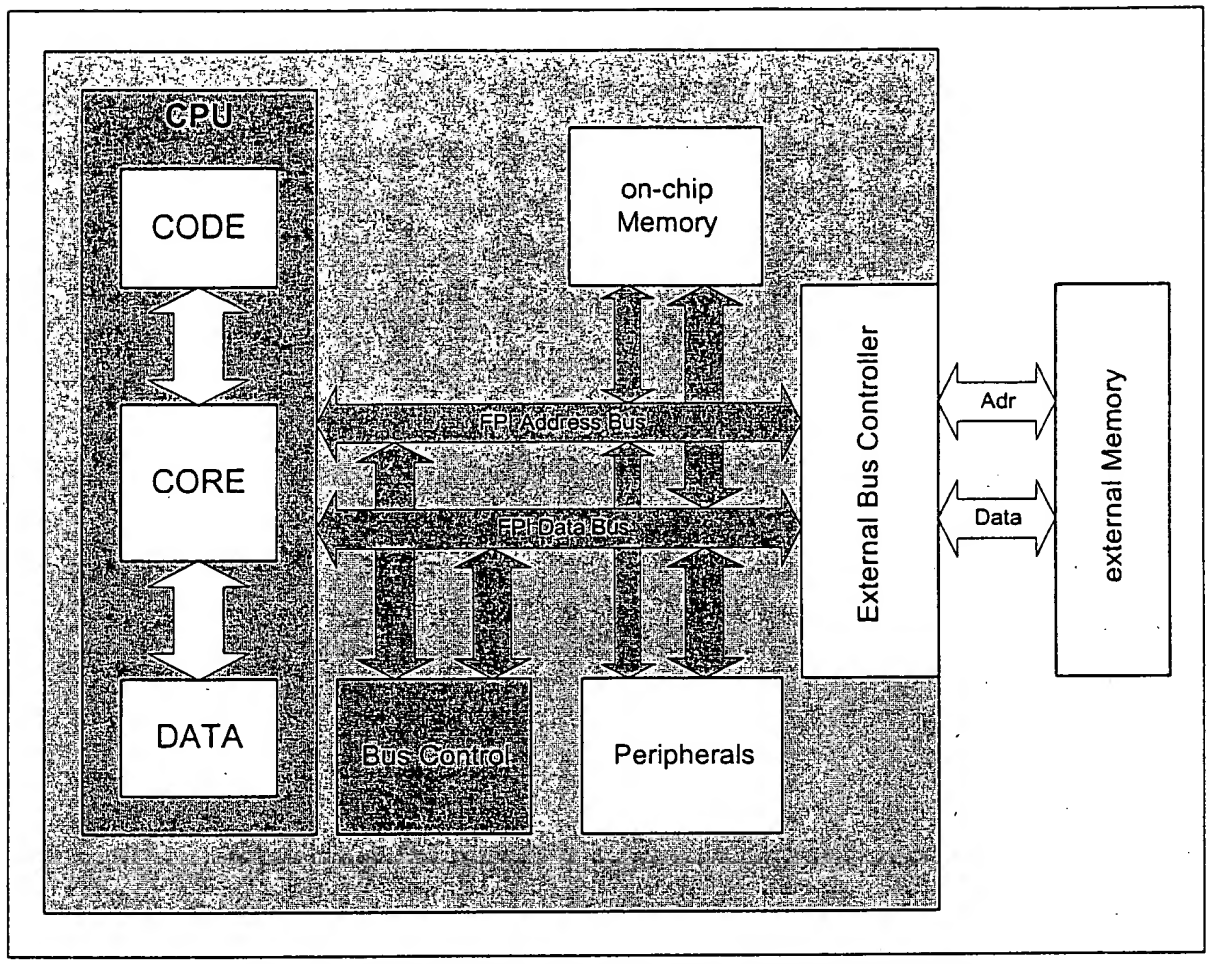

Figure 1-2: Example of a simple FPI Implementation

Although the FPI-Bus does not provide a cache coherency support, agents on the FPI-Bus may include caches. The only restriction is, that all accesses to cached areas have to go through the cache and must not bypass it (see Figure 1-3).

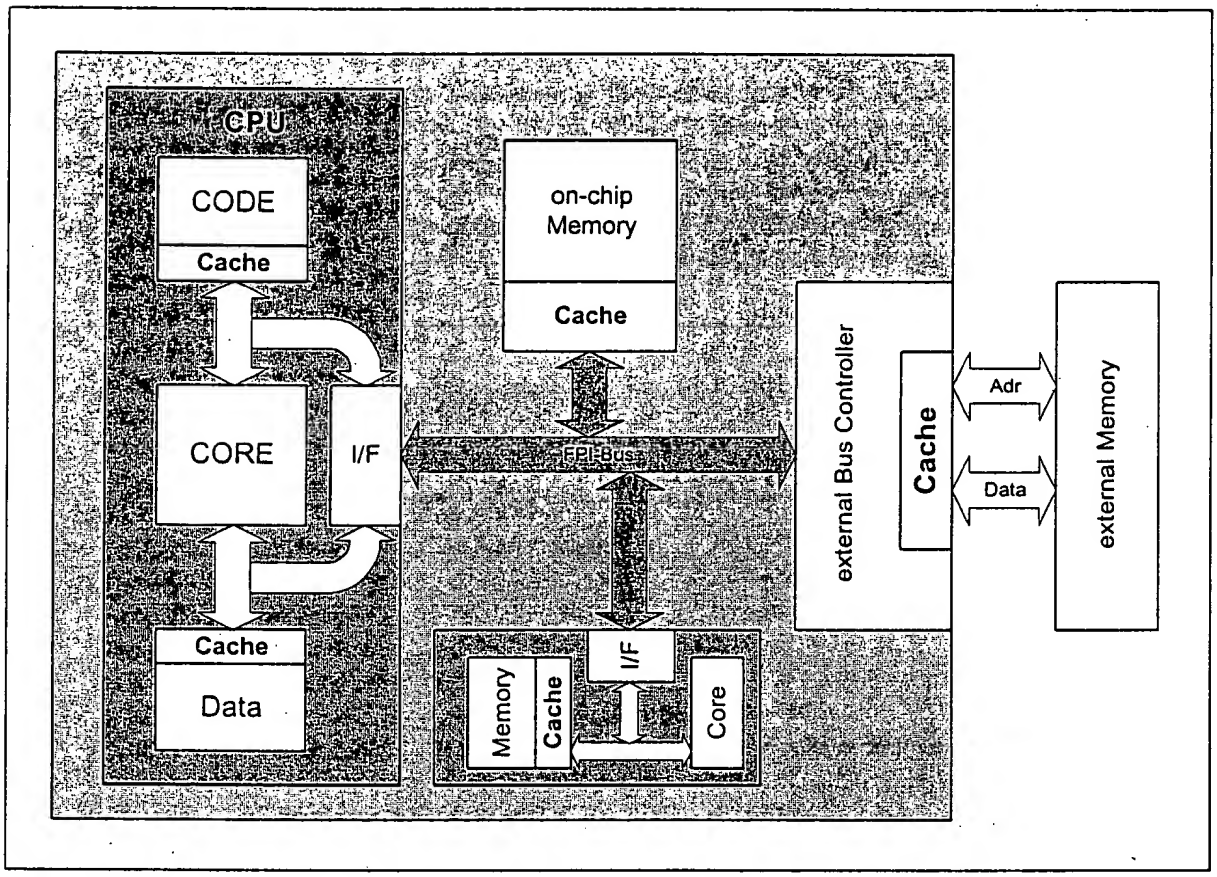

Figure 1-3: Example of caches in a FPI-Bus based system

NOTE:
In case of other bus masters sitting on the external memory bus, cache coherency on this bus has to be ensured by suitable design of this bus.

1.3 Scope of FPI-Bus Specification

The FPI-Bus specification covers the FPI-Bus protocol. It describes which set of transfers and signals at least have to be supported by FPI-Bus agents. It also lists options which may be supported by a FPI-Bus agent.

NOTE:
The FPI-Bus specification does not regulate implementation issues. In particular, it does not specify the layout of a FPI-Bus interface or define electrical parameters. This has to be specified separately depending for each implementation of the FPI-Bus.

1.4 Data Bus Width

The FPI-Bus supports data bus widths of 16, 32 and 64 bits. For simplicity the explanations and examples in this document focus on a 32-bit implementation of the bus. 16-bit or 64-bit implementations easily can be derived by changing the data bus width, the basic bus protocol will not be effected.

NOTE:
Any building of multiple accesses e.g. to transfer 64 bit data on a 32-bit data bus implementation must be done by master.

1.5 Address Bus Width

The FPI-Bus supports address bus widths of 16 to 32 bits. For simplicity the explanations and examples in this document focus on a 32-bit implementation of the bus.

For simplicity the examples show the alignment for up to 32-bit data. For 64-bit data they have to be expanded accordingly.

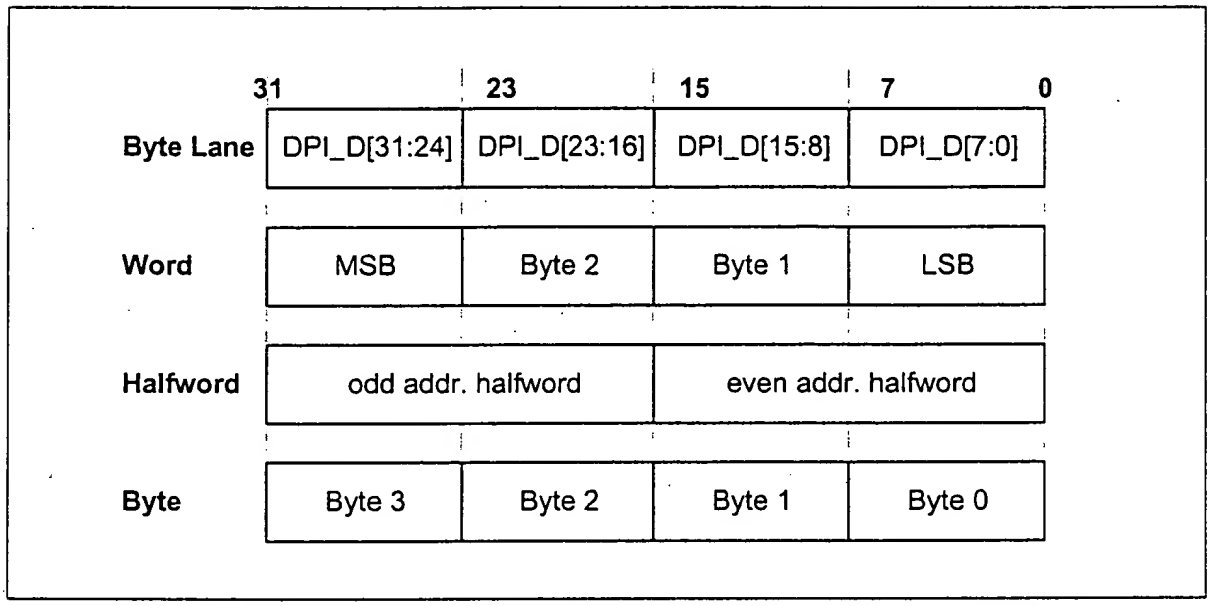

Figure 2-2: Alignment of bytes, halfword and words on FPI-Bus data lines (32-Bit implementation)

3.1 Overview of FPI-Bus Signals

The following table lists all FPI-Bus signals and some additional information about these lines. A detailed description of all signals is given in 3.2"Description of FPI-Bus Signals," on page 26.

Table 3-1: FPI-Bus Signals

Signal Name	Master	Slave	Bus Control[a]	Comments
FPI_RES_N[1:0]	I	I	I	Reset Signals
FPI_CLK	I	I	I	Bus Clock
FPI_REQ_x_N	O	-	I	Master x Request line
FPI_RES_REQ_x_N	O	-	I	Master x Split Response Request
FPI_GNT_x_N	I	-	O	Master x Grant line
FPI_BUSY_x_N	O	-	I	Pending Split Response
FPI_LOCK_x_N	O	-	I	Master x Lock line
FPI_RD_N	O	I	-	Read Control
FPI_WR_N	O	I	-	Write Control
FPI_ABORT_N	O	I	-	Abort an already started transaction
FPI_OPC[3:0]	O	I	-	Operation code
FPI_A[31:0]	O	I	I	Address bus, width as required
FPI_D[63:32] FPI_D[31:0] FPI_D[15:0]	I/O	I/O	-	Data bus
FPI_TAG[3:0]	O	I		Tag Bus (used to transfer the master ID for Split Block Transfers)
FPI_ACK[1:0]	I	O	I	Slave response code
FPI_RDY	I	O	I	Slave on data bus will be able to finish transaction in current clock cycle
FPI_SEL_y_N	-	I	O	Slave select
FPI_TOUT	I	I	O	Time-out signal
FPI_SVM	O	I	-	Supervisor Mode

Table 3-1: FPI-Bus Signals

Signal Name	Master	Slave	Bus Control[a]	Comments
FPI_NO_SPLIT_N	-	I	(O)	No Split Response

a. including bus arbiter and address decoder

The ending "_N" means that the signal is low active (V_{SS} level: signal is active, V_{CC} level: signal is not active).

3.2 Description of FPI-Bus Signals

3.2.1 System Signals

FPI_RESET_N[1],[0] (Bus Reset)

Reset forces all bus masters and slaves interfaces into idle state. In addition the peripherals can be reset. At least one reset line **FPI_RESET_N[0]** is needed. The second line **FPI_RESET_N[1]** is recommended to enable the distinction of interface reset and peripheral reset. The following table shows the coding of the reset lines (recommended reset states are print italic).

Table 3-2: Reset Encoding

FPI_RES_N[1]	FPI_RES_N[0]	Reset State
0	0	Complete Reset of all Bus Interfaces and all Peripherals
1	1	No reset
0	*1*	*Reset of all Peripherals (int. Regs. -> Reset Value)*
1	*0*	*Reset of all Bus Interfaces (no changes to int. Regs.)*

During the power-up phase **Reset** can be asynchronous, during normal operation it shall be activated synchronously to bus clock. **Reset** is always deactivated clock synchronous (with rising edge).

FPI_CLK (Bus Clock)

Bus cycle timing is referenced to the leading edges of the bus clock (**FPI_CLK**). The only exception might be the bus arbitration to enable single cycle master switch (refer to "Bus Arbitration," on page 38).

3.2.2 Bus Ownership Control Signals

All Bus Ownership Control Signals are dedicated point-to-point lines.

FPI_REQ_x_N (Bus Request)

FPI_REQ_x_N shall be driven by a master early in a bus cycle to request bus ownership from bus control for the following cycle.
All masters with active **request** lines take place in the arbitration round for the following bus cycle. If, due to a default grant, bus ownership has already been given to the master in the previous bus cycle, the master can indicates via **FPI_REQ_x_N** whether it needs the bus for one more cycle (than it takes place in the current arbitration round) or not.

Each master has a separate **FPI_REQ_x_N** line to the bus arbiter.
Therefore the number of **request** lines depends on the number of masters in the system.

A granted master has to release **FPI_REQ_x_N** immediately at the beginning of the address cycle if no further bus requests shall be issued.

NOTE:
Priority control of **request** lines is not part of this document but implementation dependent.

FPI_LOCK_x_N (Lock Bus Request)

A master that requests the bus for a certain number of consecutive, non-interruptible (chained) transfers has to issue the **Lock** line in parallel with the **Request** line. If this master is granted the bus arbiter must not perform any arbitration round until this master releases its **Lock** again.
The **Lock** line is recommended for masters that perform e.g. read-modify-write transactions.

This leads to the following kinds of requests:

Table 3-3: Locked Bus Request Encoding

FPI_REQ_x_N	FPI_LOCK_x_N	Request
0	0	Locked Bus Request
0	1	Bus Request (unlocked)
1	x	no Request

FPI_LOCK_x_N can be released without releasing the **FPI_REQ_x_N**. In this case normal arbitration goes on.

The use of locked requests is implementation specific. One should take care to avoid deadlock situation that might occur if the number of locked bus transactions is too high.

FPI_RES_REQ_x_N

A master-slave interface shall use this signal to request the bus for a pending split response. This enabled the arbiter to distinguish between normal request and pending response request.
In systems without split transactions **FPI_RES_REQ_x_N** and **FPI_REQ_x_N** shall be ored together either inside the interface or inside the arbiter (recommended).
This line is a point-to-point line from a split capable interface to the bus arbiter.

FPI_GNT_x_N (Bus Grant)

At the end of an arbitration round the bus arbiter drive one grant line (**FPI_GNT_x_N**) active to indicate to a master that its request for bus ownership has been accepted.

NOTE:
If a master did not request the bus but is granted it has to perform idle cycles.

Only one **FPI_GNT_x_N** may be active at the end of a bus cycle.

- **FPI_GNT_x_N** = 1: Master does not get bus ownership.
- **FPI_GNT_x_N** = 0: Master gets bus ownership.

Definition: A master is granted in cycle N if its dedicated **FPI_GNT_N** line and **FPI_RDY** was active in cycle N-1. It owns the data cycle starting in cycle N+1 ending with the cycle in which the transaction is terminated by activating **FPI_RDY.**

A granted master shall start operation by performing

- an address cycle or
- an idle cycle.

Earliest during this cycle **FPI_GNT_x_N** may be released again.

Each master has a separate **FPI_GNT_x_N** line from bus control.
Therefore the number of grant lines depends on the number of masters in the system.

For further details on arbitration refer to Section "Bus Arbitration," on page 38.

FPI_BUSY_x_N (optional)

A master-slave interface gathering data for a pending split response shall drive this line active. This informs the arbiter that there is a response pending.
This line is a point-to-point line from a split capable interface to the bus arbiter.

NOTE:
This line is optional and must be inplemented only in slave agents that support split transactions.

FPI_SEL_x_N (Slave Select)

This signal is a kind of chip select for the slave interface. Each slave interface has one dedicated **FPI_SEL_N** input.

Definition: A slave is selected in cycle N if its dedicated **FPI_SEL_N** line and **FPI_RDY** was active in cycle N-1. This selection is valid (for one or more cycles) until the transaction is terminated either by activating **FPI_RDY** (by the slave) or by activating **FPI_TOUT** by Bus Control.

Only the selected slave interface is active. All others have to remain in idle mode.

NOTE:
The select signal generation is not part of this specification due to the various number of different possibilities and system requirements. An example for an address decoder to generate these select signals is given in Appendix 6.5"Address Decoder (FPI_SEL_N generation),".

3.2.3 Bus Operation Signals

All bus operation signals are bussed lines which shall be driven in every active bus cycle. To avoid floating of not driven lines in case of idle cycles or bus error they will be protected with bus hold circuitries.

These bus hold circuitries are not used for bus functionality but only for power savings and lifetime extension.

FPI_READ_N, FPI_WRITE_N (Read/Write)

The read/write bus lines are driven by the granted master in address cycle to inform the selected slave that it has to provide resp. has to latch valid data in the data cycle.
(An opcode and the slave address is issued in parallel.)

Table 3-4: Read/Write Enconding

FPI_RD_N	FPI_WR_N	Description
0	0	Read-Modify-Write
0	1	Read Operation
1	0	Write Operation
1	1	NOP

Read-Modify-Write (RMW) Accesses

The FPI provides a special RMW indication for bit protection. This indication can be used by the peripheral to avoid (postpone) any internal write accesses to the addressed register during the read-modify-write.
The first read access of a RMW shall be done with read and write line active (see Figure 4-1). After the second FPI-access - a write - the internal write access shall be allowed again.

NOTE:
RMW is done in at least two cycles which shall be locked. In the first one data is read, in the second one the modified data is written back. In between no other bus agent can gain control of the bus.
A RMW accessed slave must not acknowledge with **SPLIT**.

If the master needs additional cycles to modify the data it shall perform idle cycles while the bus is still locked.

Idle Cycles

Both lines **FPI_RD_N** and **FPI_WR_N** shall be driven inactive.

FPI_ABORT_N (Abort FPI Transaction)

An already started transaction can be cancelled with this signal. **FPI_ABORT_N** must be issued in the (first) data cycle to abort the current transaction. (refer to Chapter 4.6"Error Handling," on page 74).

NOTE:
Each slave with side-effect registers shall support abort functionality. That means
a) read access => the slave should restore registers with side effects
b) write access => the slave should not take the data.

FPI_A[31:0], FPI_A[63:32] (FPI Address Bus)

Address bus lines are driven by the granted master in the address cycle. The address identifies the slave which is addressed by the bus operation. Based on this address the slave select (**FPI_SEL_x_N**) shall be generated.

NOTE:
The number of address lines to be implemented depends on system address space requirements and the address map of the implementation.

FPI_D[15:0], FPI_D[31:0], FPI_D[63:0] (FPI Data Bus)

Data bus lines are driven by the master or the slave (depending on read or write) during data cycle. The data is taken at the end of the (last) data cycle which is indicated by an active **FPI_RDY** signal.

NOTE:
The number of data lines to be implemented (16, 32 or 64) depends on the system implementation.

FPI_TAG[3:0] (FPI Tag Bus)

To enable split transfers an ID of the requester has to be transmitted on the Tag Bus. This ID is used to address the right bus agent for the split response.
Due to this scheme any master can have only one split transfer pending.

NOTE:
Due to the bus width the number of possible masters in an implementation is limited to 16.

FPI_SVM (FPI Supervisor Mode)

Distinction whether a master accessing the bus is running in supervisor mode or in user mode. This signal has to be issued in the address cycle.

- **FPI_SVM** =1: master is in supervisor mode
- **FPI_SVM** =0: master is in user mode

A slave that only supports supervisor mode for a given access, and is not accessed in that mode shall reply with ERR acknowledge code to inform the master.

FPI_NO_SPLIT_N

This signal enables or disables split transaction on the FPI-Bus.

- **FPI_NO_SPLIT_N** = 1: slave is allowed to split (**SPT** acknowledge code)
- **FPI_NO_SPLIT_N** = 0: slave must not split (no **SPT** acknowledge)

NOTE:
If **FPI_NO_SPLIT_N** is driven active during bus operation no new split transactions can be initiated but there might be some responses pending.

:FPI_OPC[3:0] (FPI Operation Code)

The opcode bus lines are driven by the granted master in the address cycle to select the type of bus transaction.

Table 3-5: Operation Code Encoding

FPI_OPC[3:0]	Identifier	Description
0000	SDTB	Single Transfer Byte (8 Bit)
0001	SDTH	Single Transfer Half-Word (16 Bit)

Table 3-5: Operation Code Encoding

FPI_OPC[3:0]	Identifier	Description
0010	SDTW	Single Transfer Word (32 Bit)
0011	SDTD	Single Transfer Double-Word (64 Bit)
0100	BTR2	Block Transfer Read (2 transfers)
0101	BTR4	Block Transfer Read (4 transfers)
0110	BTR8	Block Transfer Read (8 transfers)
0111	-	Reserved
1000	SBTR1	Split Block Transfer Request (1 transfer)
1001	SBTR2	Split Block Transfer Request (2 transfers)
1010	SBTR4	Split Block Transfer Request (4 transfers)
1011	SBTR8	Split Block Transfer Request (8 transfers)
1100	SBR	Split Block Response
1101	SBRF	Split Block Response Failure
1110	SBRE	Split Block Response End
1111	NOP	No operation

a. **NOP - No Operation.**
The granted master issues this opcode if no transaction on the bus is requested and no valid address is output. All other opcodes are accompanied by a valid address. There has to be always a master on the bus which is responsible to drive the NOP code in case of an idle bus.

b. **SDTx - Single Data Transfer.**
These opcodes are used to initiate single data read or write transfers. x= B,H,W or D specifies the data width of the transfer.
Single transfer can be split by slave answering with the **SPT** acknowledge code.
If a master gets a **SPT** acknowledge it shall release the bus and wait for one (and only one) split response.

c. **BTRx - Block Transfer Read.**
This opcode initiates a non-split block read transaction of x data items of the maximum data bus width (implementation dependent). x defines the number of items from 2 up to 8. The master will keep the bus and wait for the response. To avoid any interruption it can use a locked transaction.
This transaction might be used for burst reads from fast bus agents, e.g. on-chip page mode memories.

d. **SBTRx - Split Block Transfer Request.**
This opcode initiates a split block read transaction of x data items of the maximum data bus width (implementation dependent). x defines the number of items from 1 up to 8. Master has to release the bus after the request and wait for the response.
The tranfers might be changed into a series of single accesses if the addressed slave does not support split access.
This transaction might be used for accesses to slow bus agent, e.g. via the external bus controller.

e. **SBR - Split Response.**
If a slave supports split block transfers it drives this opcode during split answers. At the same time it asserts the ID of the former requester on the Tag Bus and the prior sent target address (it's own address) on Address Bus.
A split transfer response can be interrupt by the master itself or by bus arbitration due to a higher priority request.
If the master cannot provide the requested data one after the other it can remain it's request line active and perform idle cycles. This inserts a kind of master wait-states. If another master is granted in between the master with the open response shall continue its response if it is granted again.

f. **SBRF - Split Block Read Failed.**
To inform a former requester that a agent cannot perform a response, e.g. due to a parity error in a bridge, it sends a **SBRF** opcode together with requester tag.
The former requester has to decide to request the transfer again or do some other actions.

g. **SBRE - Split Block Resonse End**
The identifier of the last transfer of such a response is the **SBRE** opcode. It shall be sent together with the last portion of an Split Response.

FPI_RDY (Transfer Termination Indication)

This signal indicates the end of a transfer. It is normally driven active by the selected slave in the (last) data cycle. In this cycle valid information is transferred (data and/or ACK codes). To insert wait-states the selected slave can drive **FPI_RDY** inactive.
(For further information about wait-states refer to 4.2.1"Wait-state insertion," on page 57).

FPI_ACK[1:0] (FPI-Bus Acknowledge Codes)

Acknowledge code bus lines are driven by the selected slave in every data cycle. In case of idle cycles bus control has to drive the acknowledge code.

Table 3-6: FPI-Bus Acknowledge Codes

FPI_ACK[1:0]	Identifier	Description
00	NSC	**No special condition** detected for current data cycle
11	ERR	**Error**, last bus cycle aborted.
01	SPT	**Split**, modify single read transfers to split ones, acknowledges accepted split block requests
10	RTY	**Retry**, slave currently cannot respond. Master has to try later again.

The **FPI_ACK[1:0]** codes have the following meaning (see also Table 3-7):

a. **NSC - No Special Condition.**
This acknowledge code is sent by the selected slave in case of single data transfers and non-split block transactions if no error occurs.
If a slave acknowledges with **NSC** to a **SBTRx** that means that it does not support split transaction. The slave will provide the first data of the requested block together with the **NSC** acknowledge code. The requesting master has transform its block transaction to a series of single reads for the missing data.

b. **SPT - Split.**
A slave acknowledges a split transfer request with **SPT** if it supports split transactions. Otherwise it sends the achnowledge **NSC** to change the request into a series of single transfers.
To modify any single read transaction into a split transaction the slave will answer with the **SPT** acknowledge code when a master sends a single data transfer read. The requesting master has to release the bus and to wait for the answer (with opcode **SBRE**).

c. **RTY - Retry.**
If a slave generally supports the requested transaction but is due to some reason at the moment not capable to deal with it it will acknowledge with **RTY**.
If a master receives a **RTY** acknowledge code from a slave he has to repeat the current transaction. Before doing so, he shall release the bus for at least one cycles (for further details see 4.6"Error Handling,").

d. **ERR - Error.**
The selected slave terminates the current transaction with error status. (for further details see 4.6"Error Handling,").

Table 3-7: Acknowledge Codes for certain transactions

Transaction	Acknowledge Code Meaning			
	NSC	SPT	RTY	ERR
Single Data Transfer	no special condition, transfer accepted	convert single transfer to split access, master shall wait for response	busy, try later again	transfer not supported, wrong access
Split Block Transaction	conversion to single data transfers	split accepted, master shall wait for response		
non-split Block Transaction	no special condition, transaction accepted	reserved[a]		

a. must not be used

3.3 Sideband Signals

The bus specification covers issues like the bus protocol and needed signals

The FPI leaves open opportunity for implementation specific function performance enhancements via "sideband" signals. These signals are loosely defined as any signal not part of the FPI specification. They have meaning only to these agents which are connected to but not the bus itself.

3.3.1 DMA Signals

To enable fly-by DMA functionality dedicated **DMA_SEL_x_N** signals from the DMA controller to DMA destination agents must be implemented. The number of these lines is implementation dependent.

The **DMA_SEL_x_N** signal shall be issued synchronously with the FPI clock.

Table 3-8: Optional DMA Signals

Signal	Setup time		Hold time	
	typ.	worst	typ.	worst
DMA_SEL_x_N	tbd	tbd	tbd	tbd

3.3.2 Interrupt/DMA arbitration signals

For the arbitration of interrupt and DMA requests some lines are routed to any service request node. These signals can be treated as part of the sideband.

The recommended number of lines is in the range of 2 to 4.

NOTE:
An agent that implements a sideband function must exercise great care to avoid interoperability problems with FPI compliant devices that don't comprehend the function.

3.3.3 OCDS Signals

The signal **OCDS_P_SUSPEND** can be used by any On Chip Debug System (OCDS) to stop operation of each peripheral. Source will be a certain module close to the bus. If no OCDS is implemented **OCDS_P_SUSPEND** can be skipped.

This signal shall be routed in parallel to the FPI-Bus.

3.4 FPI-Bus Components

FPI-Bus components are the bus controller, master agents and slave agents.

A FPI-Bus agent needs an interface with master functionality if it is an active system module which performs read and write accesses to other modules. FPI-Bus allows to have more than one bus agent with master interface. In a multi-master configuration an arbitration logic shall be implemented to select one of several competing master interfaces.

3.4.1 Bus Controller (Bus Control)

FPI-Bus needs a bus controller for operation. It can be implemented centralized or decentralized but implementation must ensure the basic functionality:

Bus Arbitration

Due to the multi-master capability the bus arbiter has to select one of the competing masters and grant this one to the bus.

Bus arbitration in a FPI-Bus based system is done by a set of request/grant/lock lines from each potential master to the arbitration logic.

There is one speciality with the FPI-Bus, which is that there has to be always one bus master on the bus, even if no master is requesting the bus. This master is responsible for performing idle cycles which indicates an idle bus.

Again there are several strategies to select the Default Master.

NOTE:
The arbitration scheme itself is not defined by the FPI-Bus specification but is implementation dependent.
An example for an arbiter is given in Appendix 6.4"Bus Arbitration," on page 86.

Idle cycle slave functionality

During idle cycle bus control has act as default slave an provide

- active **FPI_RDY** and
- acknowledge code **NSC**.

Time-out Control

The time-out mechanism is intended for bus operations which are not completed by the selected slave with the activation of the **FPI_RDY**. Bus Control shall detect this situation and take the actions which are described in chapter 4.6.1"Time-Out Error," on page 74.

After the (synchronous) release of **FPI_RESET** the Default Master can start an FPI transfer immediately. All other masters have to wait one or two additional cycles because the arbitration round starts again in first cycle.

3.4.2 Who drives the bussed signals

The following table clearifies who drives the bussed signals in certain situations.

Assumptions:

1. Fully decoded address map: In case of an access to a non existing peripheral, the select (fpi_sel_n(0)) is given to a "default slave peripheral".
2. A slave always drives if it gets selected and OPC = $\overline{\text{NOP}}$!
3. Lets assume a design uses block select lines and not all available sfr-addresses are used. Then, the periphal must always drive appropriate FPI signals when it gets selected. Especially in case of waitstates during a read access and in case of an access to a not implemented register.
4. "x" as value specified in the tables below denotes, that the value to drive is arbitrary.
5. The signal fpi_tout is always driven by the bus controler (and therfore not a bussed signal), but it is included here to emphasize, that it is ony activated during timeout.
6. Address cycle and data cycle of the following tranfer will overlap due to the pipelined architecture of the FPI.

Table 3-9: Who drives what

Conditions	Addr Cycle		Data Cycle	
a) during reset [a]	fpi_a	0 *by bus control*	fpi_d	0 *by bus control*
	fpi_opc	NOP *by bus control*	fpi_abort_n	inactive ('1') *by bus control*
	fpi_tag	0 *by bus control*	fpi_ack	NSC *by bus control*
	fpi_rd_n	inactive ('1') *by bus control*	fpi_rdy	active ('1') *by bus control*
	fpi_wr_n	inactive ('1') *by bus control*	fpi_tout	inactive ('0') *by bus control*
	fpi_svm	inactive ('0') *by bus control*		
b) 1st cycle after reset	fpi_a	*identical to (e) default grant!*	fpi_d	*identical to (e) default grant!*
	fpi_opc		fpi_abort_n	
	fpi_tag		fpi_ack	
	fpi_rd_n		fpi_rdy	
	fpi_wr_n		fpi_tout	
	fpi_svm			
c) general (no idle)	fpi_a	x *by granted master*	fpi_d	read: x by selected slave write: x by granted master
	fpi_opc		fpi_abort_n	x by granted master
	fpi_tag		fpi_ack	NSC; SPT; RTY; ERR by selected slave
	fpi_rd_n		fpi_rdy	x by accessed slave
	fpi_wr_n		fpi_tout	inactive ('0') *by bus control*
	fpi_svm			

Table 3-9: Who drives what

d) idle cycle	fpi_a	hold previous value or zero by granted master	fpi_d	hold previous value or zero by granted master[b]
	fpi_opc	NOP by granted master	fpi_abort_n	inactive ('1') by granted master
	fpi_tag	master ID by granted master	fpi_ack	NSC by granted master
	fpi_rd_n	inactive ('1') by granted master	fpi_rdy	active ('1') by granted master
	fpi_wr_n	inactive ('1') by granted master	fpi_tout	inactive ('0') *by bus control*
	fpi_svm	inactive ('0') by granted master		
e) grant and no request (default grant)	fpi_a	***identical to (c) general or (d) idle cycle, decision by default master!***	fpi_d	***identical to (c) general or (d) idle cycle according to address cycle***
	fpi_opc		fpi_abort_n	
	fpi_tag		fpi_ack	
	fpi_rd_n		fpi_rdy	
	fpi_wr_n		fpi_tout	
	fpi_svm			
f) wait-state	fpi_a	***identical to (c) general !***	fpi_d	read: hold previous value or x by default slave write: x by granted master
	fpi_opc		fpi_abort_n	***identical to (c) general !***
	fpi_tag		fpi_ack	
	fpi_rd_n		fpi_rdy	
	fpi_wr_n		fpi_tout	
	fpi_svm			

Table 3-9: Who drives what

g) timeout	fpi_a	*identical to (c) general !*	fpi_d	read: hold previous value or x by bus control write: x by granted master
	fpi_opc		fpi_abort_n	x by granted master
	fpi_tag		fpi_ack	NSC by bus control
	fpi_rd_n		fpi_rdy	active ('1') by bus control
	fpi_wr_n		fpi_tout	active ('1') *by bus control* [c]
	fpi_svm			
h) access to non existing peripheral;	fpi_a	*identical to (c) general !*	fpi_d	read: hold previous value or zero *by selected (default) slave* write: x *by granted master*
	fpi_opc		fpi_abort_n	x *by granted master*
	fpi_tag		fpi_ack	ERR *by selected (default) slave*
	fpi_rd_n		fpi_rdy	active ('1') *by selected (default) slave*
	fpi_wr_n		fpi_tout	inactive ('0') *by bus control*
	fpi_svm			

Table 3-9: Who drives what

i) access to non existing register (address) in a peripheral	fpi_a	*identical to (c) general !*	fpi_d	read: hold previous value or zero *by selected slave* write: x *by granted master*
	fpi_opc		fpi_abort_n	x *by granted master*
	fpi_tag		fpi_ack	ERR *by selected slave*
	fpi_rd_n		fpi_rdy	active ('1') *by selected slave*
	fpi_wr_n		fpi_tout	inactive ('0') *by bus control*
	fpi_svm			

a. Reset is released with the rising edge of clock. The time from activation of reset to the rising edge after reset has been deactivated is considered case (a). The following clock cycle is considered case (b).
b. The Masters that is owner of the data cycle (was granted in the previous cycle).
c. The owner of the data cycle (granted master of the previous transfer) has to treat a time-out terminated transfer similar to a ERR terminated transfer. At least it shall terminate the transfer.

NOTE:
The difference between (h) and (i) is, the select. In case (h) no existing peripheral is selected. Therefore the default peripheral drives the bussed signals. In case (i) an existing peripheral is selected and it has to drive signals as listed in the tabe.

Debug Support

The Bus Controller has to trace all information given on the FPI bus. In case of error debug information shall be stored within Bus Controller internal registers and an interrupt request shall be asserted. Error in this case means **ERR** acknowledged transactions and **TIMEOUT** terminated transactions.

In order to obtain the data, some FPI signals must sampled at the end of an address cycle while others shall be collected in the corresponding data cycle.

Information stored in the address cycle:

- FPI_A[31:0]
- FPI_TAG[3:0]
- FPI_OPC[3:0]
- FPI_RD_N
- FPI_WR_N

Information stored in the data cycle:

- FPI_D[31:0]
- FPI_ACK[1:0]
- FPI_RDY
- FPI_ABORT
- FPI_TOUT

Debug information will be available only for first error. A second error will not rewrite the error registers.

In order to prevent accidental access, read and write of the error registers shall be possible in supervisor mode only (if available). A read of all error registers shall enable a new trace.

3.4.3 Master Agent

A master interface has to provide the following functionality:

Reset

Any activation of the reset signal shall force the master interface into a reset state until the reset signal has been released. Due to the two reset lines four different reset states are possible. These are described in more detail in the Section 3.2.1"System Signals," on page 26.

Idle Mode

All masters that are not granted have to be in idle mode.
An idle master shall keep all its bus drivers deactivated. To get bus ownership a master has to set the request signal (**FPI_REQ_N**) active.

Request/Grant

A master is requesting the bus by activating its request line. To request for consecutive locked bus transfers it has to activate **FPI_LOCK_N** in parallel.
A master is winner of the bus arbitration if its grant line is activated by bus control. It shall start an address cycle an idle cycle in case of grant and **FPI_RDY** active.

Address Cycle

Every bus transfer starts with an address cycle. In this cycle the information about the transfer type and the target slave is given. The master has to drive
- **FPI_OPC[3:0]**,
- **FPI_A[31:0]**,
- **FPI_TAG[3:0]**,
- **FPI_RD_N**,
- **FPI_WR_N** and
- **FPI_SVM.**

A master that wants to perform only one bus access shall release its request/lock line in the address cycle.

Due to the bus pipeline the address cycle may be executed in parallel to the data cycle of the previous transfer. If this data cycle is not terminated (FPI_RDY inactive) the address cycle must be repeated in the following cycle.

Idle Cycle

Idle cycle is a special kind of address cycle that does not initiate any bus transaction but indicates an idle bus.
A master which is granted without request or is granted and cannot perform a bus transfer shall perform idle cycles. It has to drive:
- opcode **NOP**
- **FPI_RD_N** inactive,

- **FPI_WR_N** inactive.

Data Cycle

write:
In a write data cycle the master has to
- drive **FPI_ABORT_N**,
- provide data on **DPI_D[31:0]**,
- get acknowledge code from the slave on **FPI_ACK[1:0]** and
- evaluate the **FPI_RDY** signal.

If **FPI_RDY** is not active (wait-state insertion) the data cycle must be repeated in the following cycle.

read:
In a read data cycle the master has to
- drive **FPI_ABORT_N**,
- latch data from **DPI_D[31:0]** (provided by slave),
- get acknowledge code from the slave on **FPI_ACK[1:0]** and
- evaluate the **FPI_RDY** signal.

If **FPI_RDY** is not active (wait-state insertion) the master must neither latch data nor acknowledge code. The data cycle must be repeated in the following cycle.

Due to the bus pipeline the data cycle may be executed in parallel to the address cycle of the following transfer

Time-out

A master (owner of the data cycle) which sees an active time-out signal (**FPI_TOUT**) shall abort the current transaction and release the bus in the following cycle.
For further information refer to Section 4.6"Error Handling,".

Abort

A master which wants to abort an already started transaction can do this by driving **FPI_ABORT_N** active in the (first) data cycle of the transfer. This shall force a slave to cancel the transaction.

3.4.4 Slave Agent

A FPI-Bus agent with slave functionality needs an interface, which is addressed via read and write transfers by master agents.

Each slave interface is selected via a dedicated slave select signal (**FPI_SEL_x_N**). This select signal is normally decoded from the high order address bits of the address issued by the granted master. An example for the select generation is given in Appendix 6.5"Address Decoder (FPI_SEL_N generation),". A slave is only activated if its **FPI_SEL_x_N** is active and **FPI_RDY** is high.

A slave interface has to provide the following functionality:

Reset

Any activation of the reset signal shall force the slave interface into a reset state until the reset signal has been released. Due to the two reset lines four different reset states are possible. These are described in more detail in the Section 3.2.1"System Signals," on page 26.

Idle Mode

All slaves that are not selected shall be in idle mode.
An idle slave shall keep all its bus drivers deactivated.

Address Cycle

Every bus transfer starts with an address cycle. In this cycle the information about the transfer type and the target slave is given. The slave has to latch the information on:
- **FPI_OPC[3:0]**,
- **FPI_A[31:0]**,
- **FPI_TAG[3:0]**,
- **FPI_RD_N**,
- **FPI_WR_N** and
- **FPI_SVM.**

Idle Cycle:
no actions

Data Cycle

The slave controls the data cycle of the transfer. Therefore it has to provide the following information:
- transfer acknowledge code on **FPI_ACK[1:0]** and
- transfer termination information on **FPI_RDY**.

write:
In a write data cycle the slave has to
- transfer acknowledge code on **FPI_ACK[1:0]**,
- transfer termination information on **FPI_RDY**.
- latch the data on **DPI_D[31:0]**,

If **FPI_RDY** is not active (wait-state insertion) the data cycle must be repeated in the following cycle.

read:
In a read data cycle the slave has to
- latch data from DPI_D[31:0] (provided by slave)
- transfer acknowledge code on **FPI_ACK[1:0]** and
- evaluate the **FPI_RDY** signal.

If **FPI_RDY** is not active (wait-state insertion) the master must neither latch data nor acknowledge code. The data cycle must be repeated in the following cycle.

idle cycle:
The slave has to evaluate the **NOP** opcode and must not perform any action.

Due to the bus pipeline the data cycle may be executed in parallel to the address cycle of the following transfer

Time-out

A selected slave which sees an active time-out signal (**FPI_TOUT**) shall abort the current transaction and release the bus in the following cycle.
For further information refer to Section 4.6"Error Handling,".

Abort

If a slave gets an active **FPI_ABORT_N** in in the first data cycle of a transfer is has to
- abort the current transaction, restore (peripheral) internal registers if possible and
- drive **FPI_RDY** active in the following cycle if wait-states were inserted.

3.4.5 Master-Slave Agents

A bus agent that provides master as well as slave functionality is called master-slave interface.

It is recommended to implement a master-slave interface for every master capable peripheral due to the minimal hardware overhead and the enhanced functionality.

3.5 Default Master

The Default Master will be granted if no bus request is active. It has to perform idle cycles to indicate an idle bus or any other transaction.

There are several schemes possible to select a bus master for this purpose:

- the last bus master might get his GNT line kept active as long there is no other request activated
- one selected implementation dependent master(e.g. CPU) can be the default master, getting the default grant
- the bus controller itself might provide the NOP, being the implicit default master

---

NOTE:
The default master has the advantage of being able to start a transaction immediately without request. This saves at least one cycle.

3.5.1 Examples of Routing FPI-Bus Signals

Figure 3-1 shows FPI-Bus signal lines routed between FPI-Bus components, a master interface, a slave interface and the central bus control.

NOTE:
It is assumed that the bus arbitration and timout control is done centralized in the bus control which is recommended. The data bus in this example is a 32 bit implementation.

FPI_RESET_N[1:0] and **FPI_CLK** are system signals connected to every bus component.

The bus ownership control lines **FPI_REQ_A_N**, **FPI_LOCK_A_N** and **FPI_GNT_A_N** are point-to-point lines from master A to the bus arbiter (bus control).

The **FPI_TOUT** signal is driven by the timeout control (bus control) and routed to every master and every slave.

FPI_RD_N, FPI_WR_N, FPI_ABORT_N, FPI_OPC[3:0], FPI_A[31:0], FPI_TAG[3:0], FPI_D[31:0], and **FPI_ACK[1:0]** are driven by more than one source (bused signal lines). The figure concentrates on the connection between master A and slave B.

The maximum width of the address bus (normally **FPI_A[31:0]**) depends on the system memory space of the implementation. The upper address bits are mapped to the slave select signals. The lower bits address slave internal memory locations, like registers, FIFOs or RAM.
A master has to be connected to all implemented address lines, a slave only to those address lines which are needed for the slave internal decoding.

The maximum width of the data bus is determined by the largest data type (byte, halfword, word or double-word) which shall be transported over FPI-Bus during normal transfers. The minimum number is determined by the data size of the core (16-bit core => D[15:0], 32-bit core => D[31:0], 64-bit core => D[63:0]).

It is possible to reduce the number of data lines routed to small data size peripherals. However, it has to be insured that the addresses for the peripheral registers then will be located on addresses which are *modulo(*max. data bus width*)*. For example, if an 8-bit peripheral shall be connected only to D[7:0] of a 32-bit bus, its register addresses must be located on word boundaries.

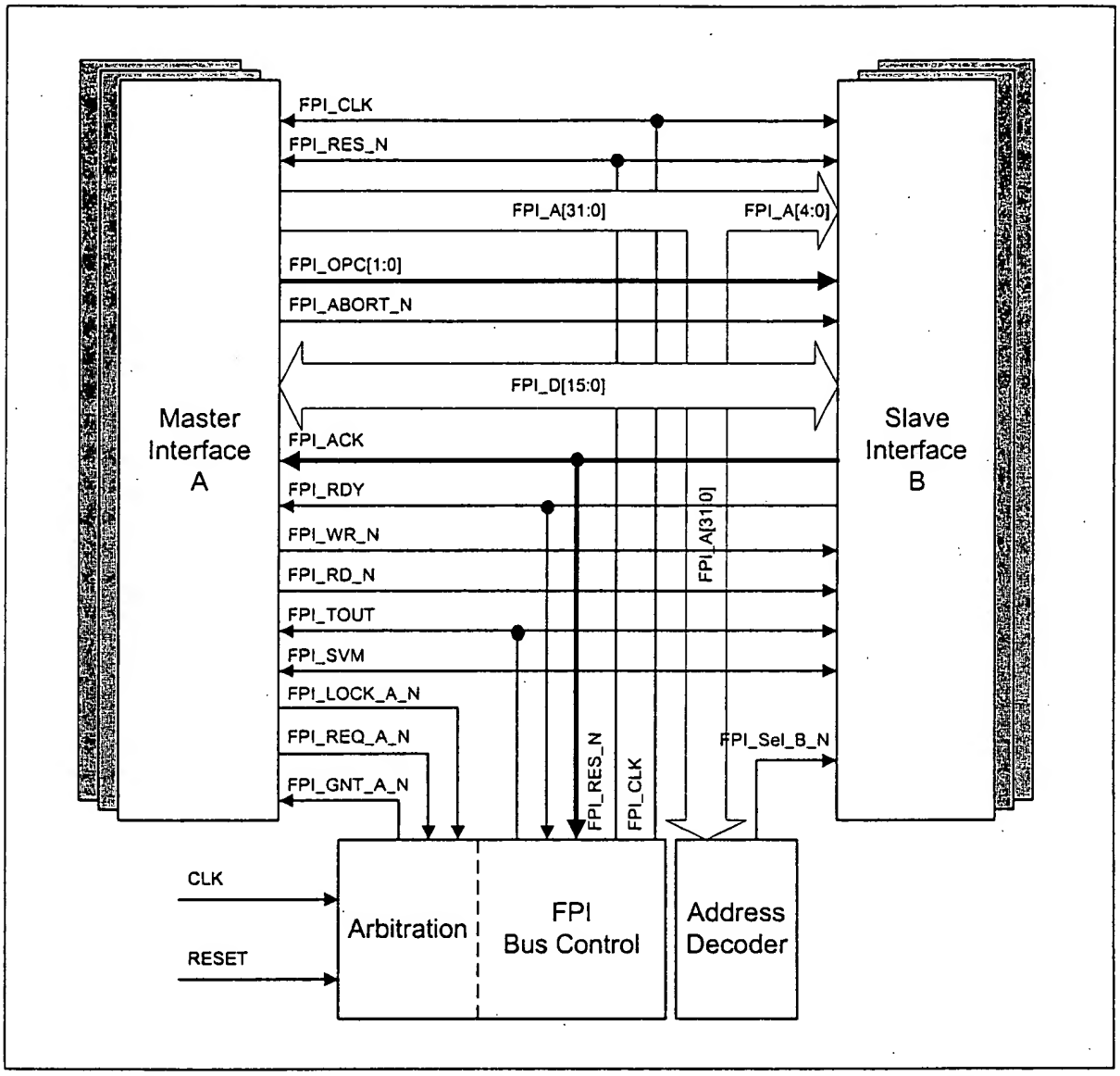

Figure 3-1: Example of routing between master and slave interface (32 bit implementation assumed)

3.6 Power Management

This specification does not cover power management measures. The following proposals may be implemented:

- switch off the bus clock whenever the bus is idle
- switch off clocking of interfaces that are not active

4.1 Single Data Transfer

The timing diagram in Figure 4-1 shows the basic single data transfers defined on FPI-Bus: read, write and read-modify-write.
Read and write transactions take always 1 cycle (without wait-states) and a read-modify-write access takes 2 cycles.

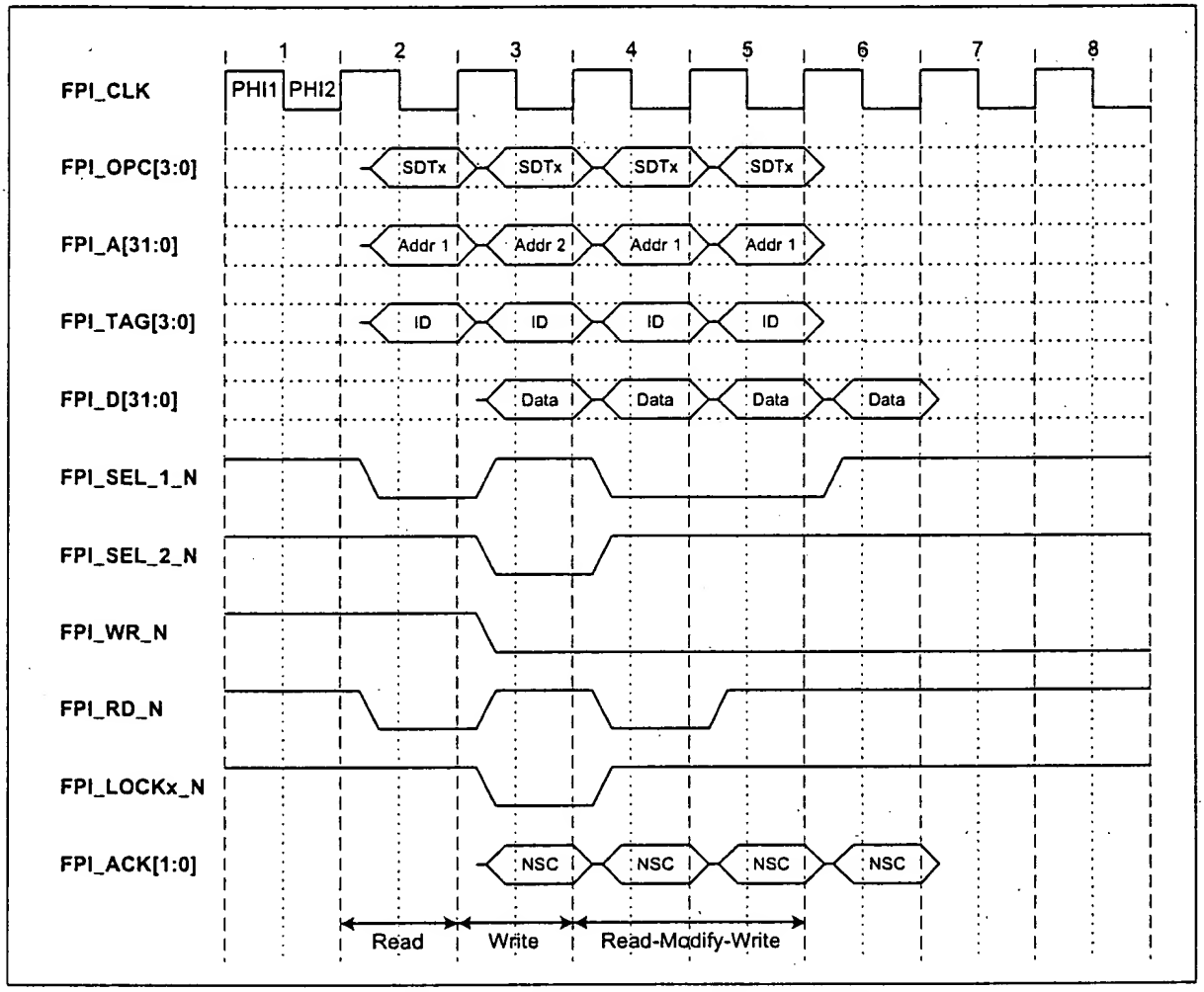

Figure 4-1: Basic Single Data Transaction

Assumptions: **FPI_RDY** is active all the time (no wait-states).

Cycle 2: This is the address cycle of the transfer.
The granted master starts a read data transfers driving the opcode for a single data transfer and setting the read and write control lines to appropriate levels.

The address of the source agent is issued on the address bus and the master ID on the tag bus.
The slave is selected by driving the appropriate select line (**FPI_SEL_1_N**) active in this cycle (by address decoder).

Cycle 3: This is the data cycle of the read transaction started in previous cycle.
The slave selected in cycle 2 drives the requested data onto the data bus, drives **FPI_RDY** active and issues the acknowledge code (**NSC**).

Due to the pipelining the address cycle of the following transaction (a write) is performed in parallel and a new slave is selected by its **FPI_SEL_2_N** line.

Cycle 4: The data cycle of the write is performed.

In parallel the granted master issues the address cycle of the next access (read-modify-write). This access consists of two bus transfers - one read and one write - which have to be locked by the master to avoid any interruption. In the first transfer the data is read from the slave agent. This read is set up in this cycle. Pay attention to the active read and write line. This indicates the read-modify-write. Slaves which cannot deal with this shall treat this access as a normal read.
FPI_SEL_N_1 is activated again.

Cycle 5: The data cycle of the read is performed. The master gets the data from the slave and can start the modification, e.g set bits, clear bits.

In parallel the address cycle of the second transfer (the write-back) is performed.
If the master does not want to perform another transaction it can release its request.

Cycle 6: Due to the released request another master gained control and perform an address or idle cycle (not shown in the figure).

In parallel the data cycle of the write-back is done. The master (the owner of the data cycle) provides the modified data and the slave latches this data at the end of the cycle.

NOTE:
A Read-Modify-Write transaction takes at least two cycles that shall be locked. This is ensured by a locked bus request.
If the master needs additional cycles to modify the data it could perform idle cycles while the bus is still locked.

A Single Data Transfer Read can be split by slave if it supports Split Transactions. Figure 4-2 shows such a scenario.

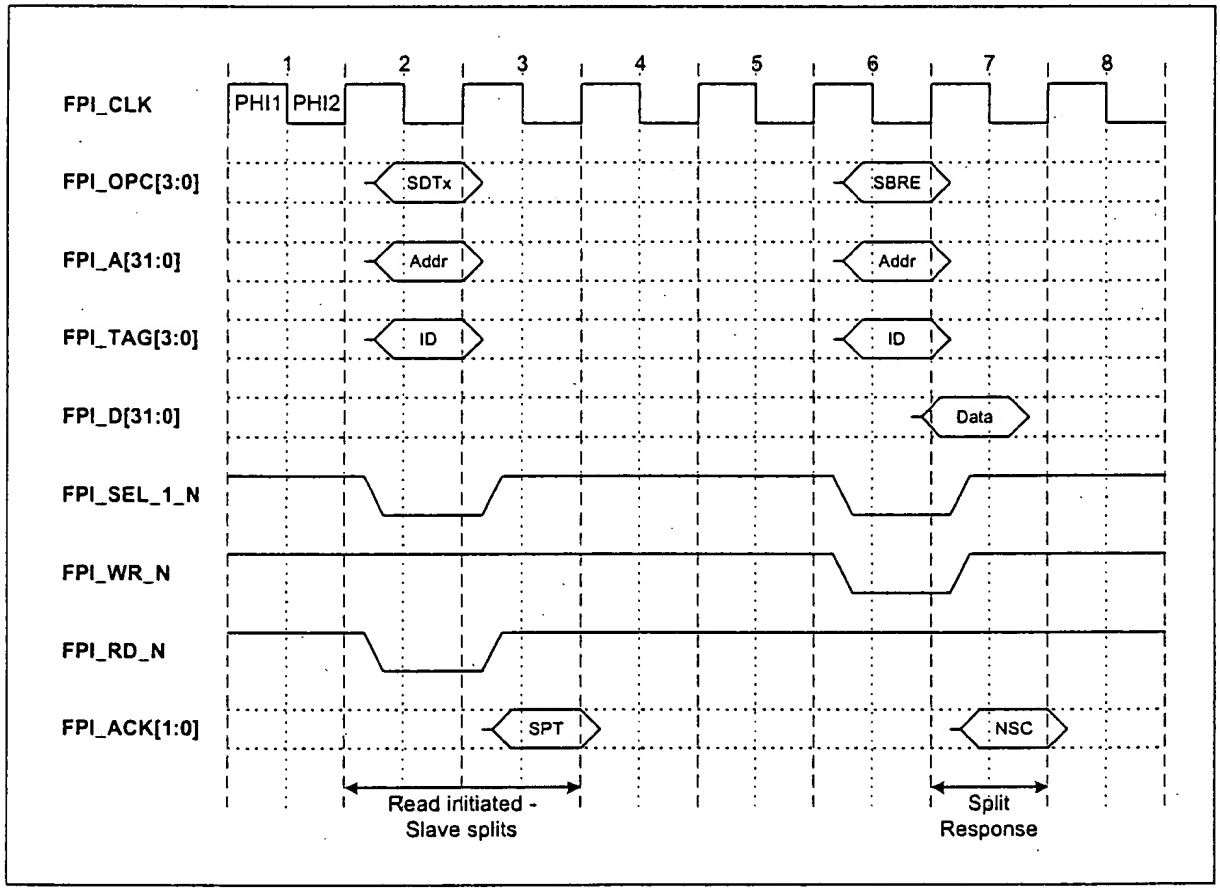

Figure 4-2: Single Data Transfer Split

Cycle 2:	A Single Data Transfer Read is initiated in the address cycle.
Cycle 3:	The Slaves drives **FPI_RDY** active and answers with **SPT** acknowledge code. The master has to wait for the split response and it shall release the bus (if it is still granted). (**FPI_RD_N** is driven high by the default master which does not want to perform any transaction.)
Cycle 4...5:	Any other bus action takes place.
Cycle 6:	The former slave has gathered the requested data and grabs the bus as master. It performs the address cycle of the split response by driving its own address (to prevent another slave from being selected), the ID of the requester (former master), the **SBRE** opcode and **FPI_WR_N**.
Cycle 7:	In this cycle the requested data is transferred. In parallel the address cycle of the next transaction can be performed. (**FPI_WR_N** is driven high by the default master which does not want to perform any transaction.)

The sequence on the bus are similar to a Split Block Request except that the decision if split or not is done by slave not by master.

4.2 Single Data Transactions with Wait-states

4.2.1 Wait-state insertion

An active **Ready** signal (**FPI_RDY**) at the end of a cycle indicates that the current transfer is terminated. Valid information will be transferred (data and/or ACK codes).

A slave can insert wait-states if needed by driving **FPI_RDY** inactive during the data cycle. This has the following consequences for the bus protocol:

- The master (owner of the current data cycle) has to evaluate this information and
  - repeat the data in the following cycle if the current transfer is a write or
  - do not latch any (invalid) data if the current transfer is a read.
- The granted master (owner of the address cycle) has to repeat the address cycle in the following cycle.

- The bus arbiter must not perform a new arbitration round in the current cycle.

Figure 4-3 gives some examples for wait-state insertion.

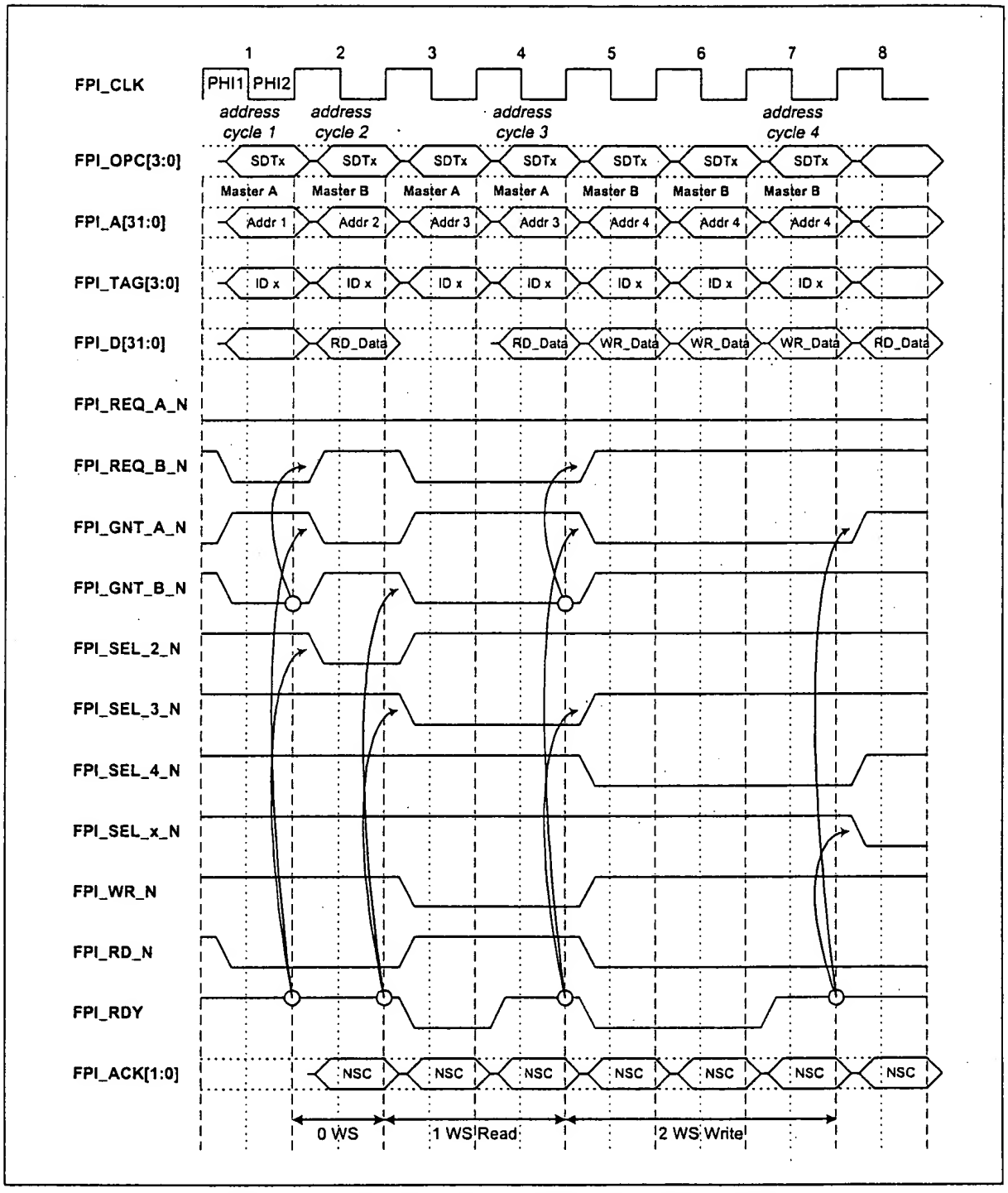

Figure 4-3: Single Data Transfer with Wait-states

4.3 Single Data Transfer Error

If a slave principally cannot execute the required transaction (e.g. wrong data size, access in supervisor mode), it shall respond with an error (**ERR**) acknowledge code.

Figure 4-4 gives an example for this scenario (For further details on error handling refer to 4.6"Error Handling,").

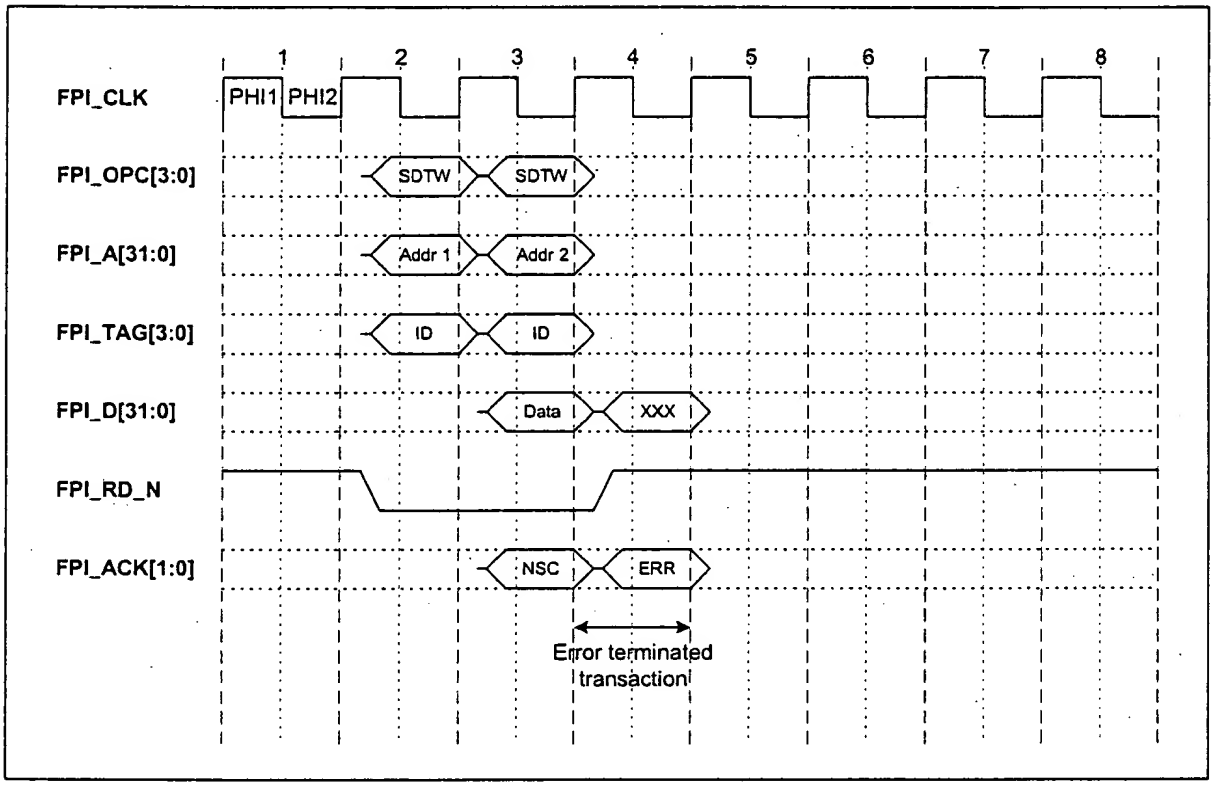

Figure 4-4: Error terminated Single Data Transfer

Cycle 2: The granted master starts a read operation with the address cycle.

Cycle 3: The master (owner of the data cycle) gets the data.

In parallel another read transfer is initiated.

Cycle 4: The master (owner of the data cycle) gets an acknowledge code **ERR** from the slave.
In this example (read transaction) the slave cannot provide valid data. In case of write transactions the data is not latched by slave.

NOTE:
If a slave cannot accept a transaction at the moment, but can generally perform the requested transaction it shall respond with an **RTY** acknowledge code.

4.3.1 Single Data Transfer Retry

If a slave can perform the required transaction generally but not at the moment, e.g. due to a pending split response, it shall respond with the **RTY** acknowledge code.
If a master gets a **RTY** acknowledge code it shall release the bus for at least one cycle.

For further details refer to Section 4.6.4"Retry (RTY) Acknowledge Code," on page 77.

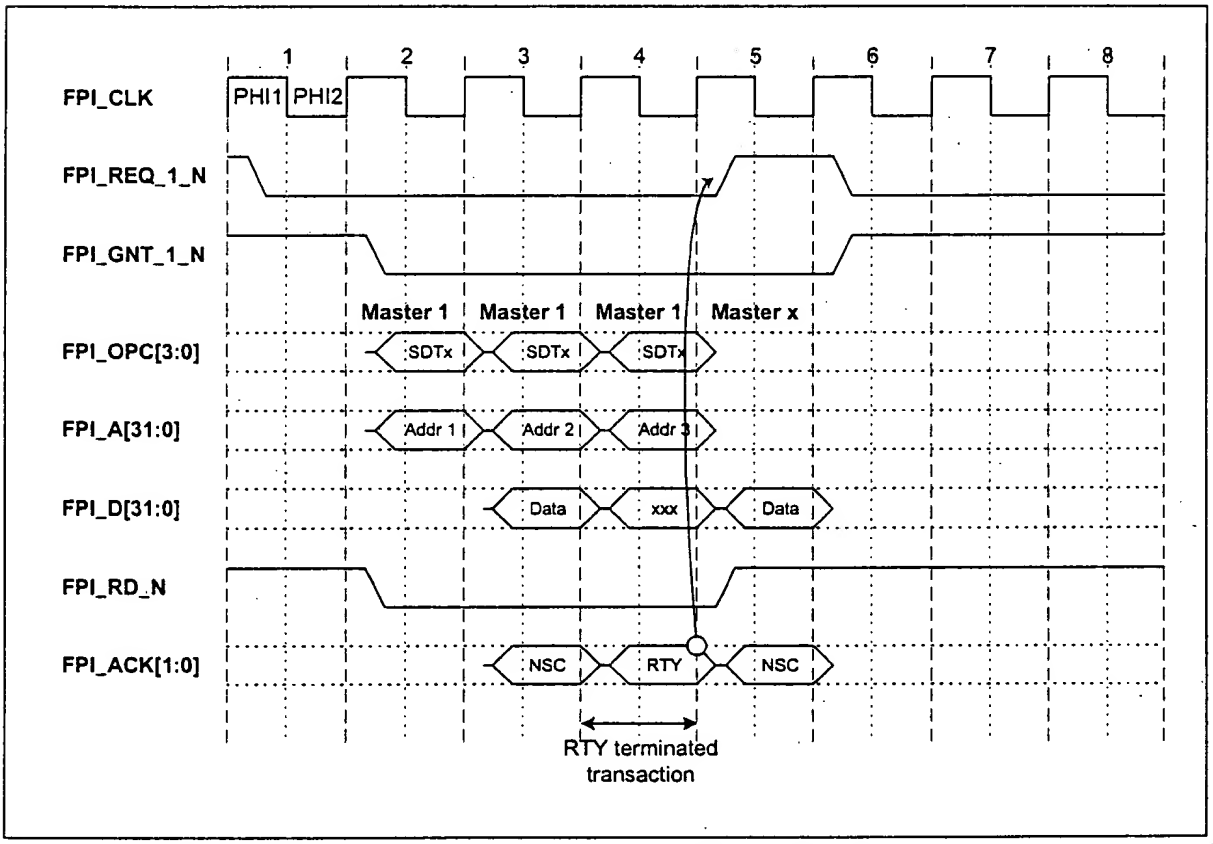

Figure 4-5: Single Data Transfer with RTY Acknowledge Code

Cycle 2: The granted master starts to do some consecutive read Single Data Transfers.

Cycle 4: The master gets an acknowledge code **RTY** from the slave.
The master shall release the bus for at least one cycle.

Cycle 5: The master gets the data from the 3rd transfer (different slave).
The master releases its request and another master may be granted.

Cycle 6: Another master is granted and the master that got the **RTY** issues its request again.

NOTE:
In cycle 5 an out of ourder execution of transfers occurs. To avoid this **FPI_ABORT_N** could be used.

4.3.2 Abortion of started transactions

To ensure the sequence of consecutive bus transfers in case that the slave acknowledges one access with **RTY** the master has the possibility to abort a transaction. All slaves with side-effect registers shall support the abort functionality for read accesses. For write accesses every slave has to support abort functionality.
An example is given in Figure 4-6.

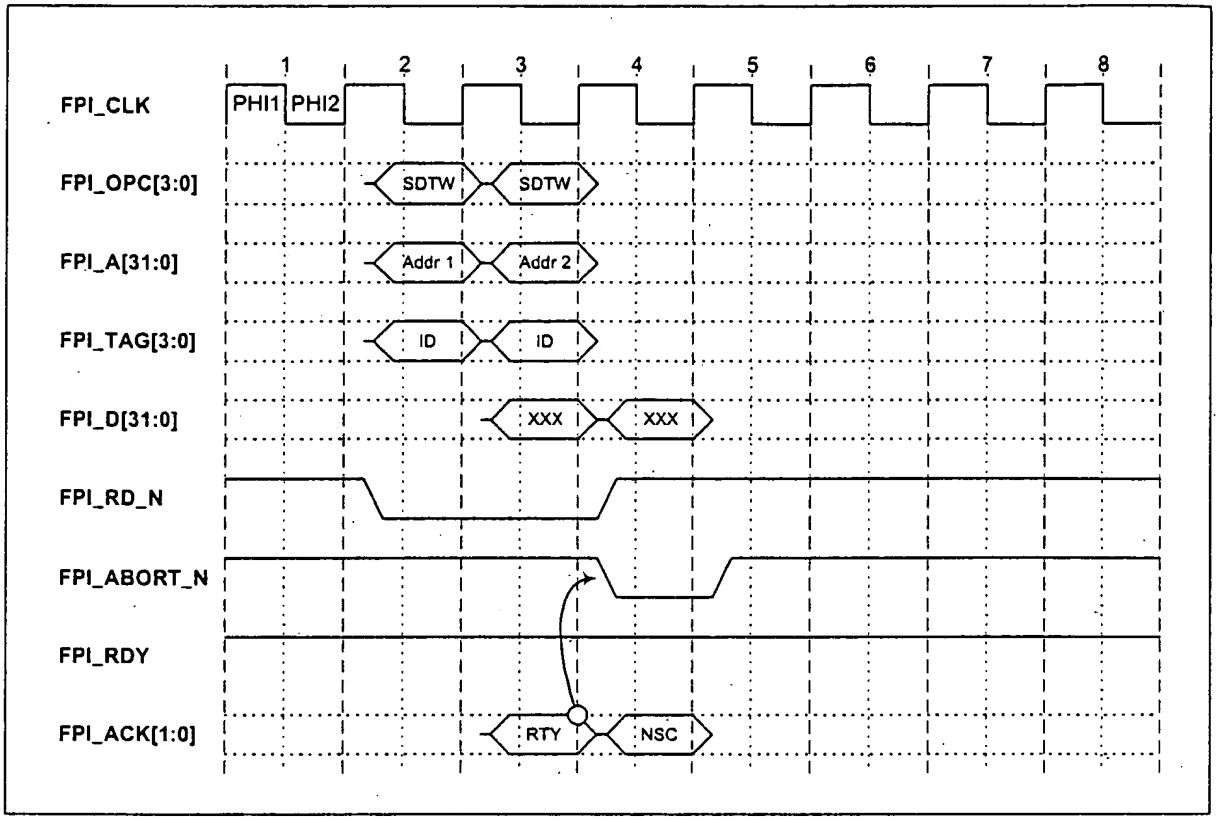

Figure 4-6: Abortion of transactions

Cycle 3: The master gets the acknowledge code **RTY** while the address cycle of the following transfer is already issued.

Cycle 4: The master drives **FPI_ABORT_N** active to enforce the slave to abort the current transfer.
The slave cancles the transfer. In case of read it may provide the requested data but the master will not use it due to the abort. Internally it has to restore the status as if the access did not happen (in case of side-effect registers). In case of write the slave must not take the data.

The master can try to repeat the consecutive accesses later again.

4.4 Block Read Transactions

Two different Block Read Transactions are defined on FPI-Bus:

- Split Block Read Transactions and
- Block Read Transactions (non-split).

4.4.1 Split Block Read Transaction

As the name already says, this kind of read transaction is split into two transfers: The response is separated from the request transfer by at least one bus cycle to turn the bus to the slave. Depending on the latency it takes the slave to provide the data this time might be extended. During this time the bus is not blocked and can be used by other agent for transactions.

- During the request information about the target address, the master ID and the number of data items is sent from the master to the selected slave. The request sequence consists of one cycle.
- During the response the former slave gains control of the bus as master and sends the requested number of data items to the former requester identified by its ID. This answer is interruptible if it is not protected by **Lock**.

The maximum number of data items per each block transaction is limited to 8 (refer to Section ": FPI_OPC[3:0] (FPI Operation Code),").

Interruption of Split Block Transfers

The data phase (response) may be interrupted by other transactions. In order to allow the receiving agent properly to accept the next data after an interruption both the initiating master as well as the transmitting slave have to keep track on the data sent, so that they will recognize an interruption, a resumption and the end of the transfer and be able to initiate or receive a new block transfer.

NOTE:
For response end recognition a special opcode **SBRE** is sent during the last data transfer.

Multiple Split Block Transfers.

There may be multiple split block transfers open at a time, depending on the number of split access capable slaves on the bus. Even one slave device may have more than one request open. However, the performance gain is expected to be very small, compared to the additional complexity in this slave.

Therefore it is expected that most split capable slaves will allow only one transaction open at a time (implementation dependent):

If a slave receives another Split Block Read Request while it already has one open, two cases have to be distinguished:

- The master ID is the same as the one of the pending response:
  - The target address is different from the previous one:
    In this case the slave will send a RTY acknowledge code, indicating that it cannot accept the request at the moment.
  - The target address is the same:
    In this case the slave has to cancel the pending tranfer and serve the latest request. This is for example necessary to insure a predictable interrupt response times.
- The master ID is different from the one of the pending response:
  - In this case the slave will send a RTY acknowledge code, indicating that it cannot accept the request at the moment.

An example for a split block read transaction is shown in Figure 4-7.
It is assumed that the split response is not interrupted.

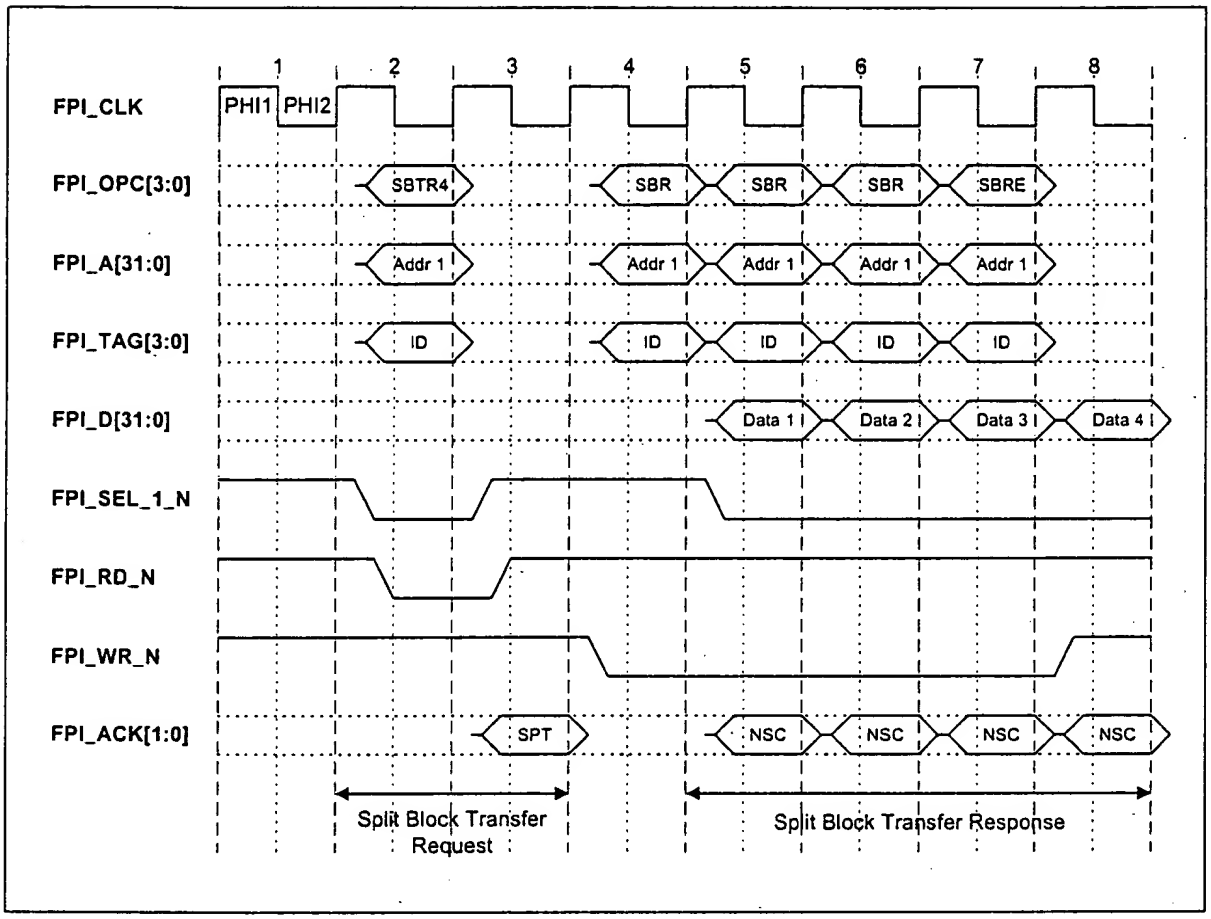

Figure 4-7: Split Block Read Transaction

Cycle 2: The granted master gains control over the bus and issues the Split Block Transfer Request of 4 data items (**SBTR4** opcode) and releases the bus.

Cycle 3: The addressed slave acknowledges with **SPT**.
Another other bus operation starts with the address cycle.

Cycle 4: The former slave of the split block transfer request is granted as master.
It starts the split response with the address cycle issuing the **SBR** opcode, the former requester ID and its own address. At the same time it drives **FPI_WR_N** active.
The transfers are similar to 4 consecutive write transfers but with different opcode and ID. They may be locked.

Cycle 5: The first data issue is sent.

Cycle 6: The second data issue is sent.

Cycle 7: The third data issue is sent together with the **SBRE** opcode. This informs the slave that only one more item will follow. After that the response is terminated.

Cycle 8: The last data item is sent and in parallel a new address cycle of another transfer or an idle cycle is performed.

NOTE:
The data width of the transferred data is always the maximum data bus width (implementation dependent).

4.4.2 Retry of Split Block Read Transactions

A retry of Split Block Read Request only can occur if a bus master is requesting a block from a slave which has already its maximum number of requests (normally 1) open (not finished).
As the example in Figure 4-8 shows the principle is the same as for single cycle retries.

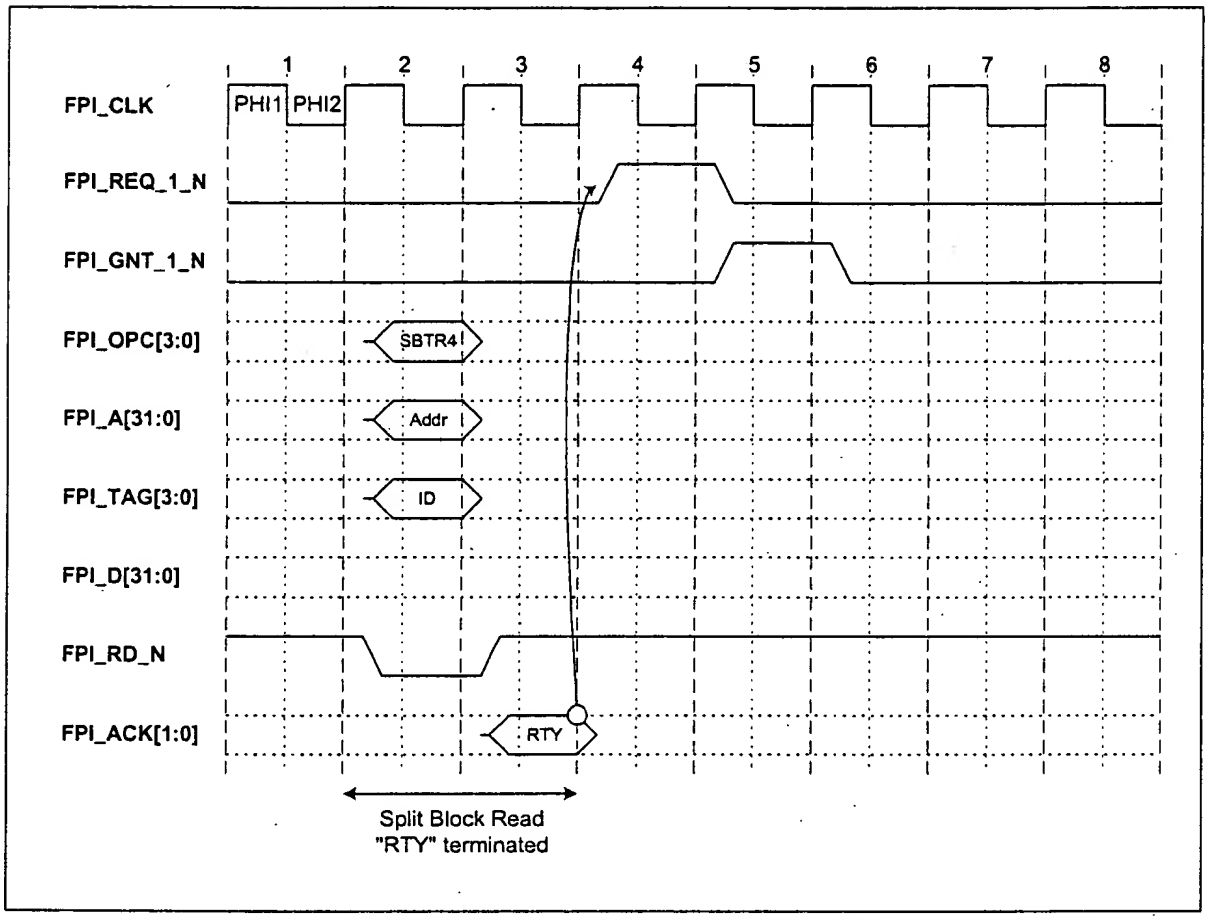

Figure 4-8: Split Block Transfer Request with RTY Acknowledge Code

Cycle 2: A Split Block Read Transaction is initiated by issuing the split transfer request.

Cycle 3: The masters gets the acknowledge code **RTY**. The slave cannot accept the request at the moment and it asks the master to repeat the transfer later again. If the master is still granted it shall release the bus for at least one cycle.

4.4.3 Split Block Transaction Conversion

A slave which cannot perform split block transfers shall convert the split block read to consecutive single accesses. This can be done by answering with **NSC** acknowledge code instead of **SPT**. An example is shown in Figure 4-9.

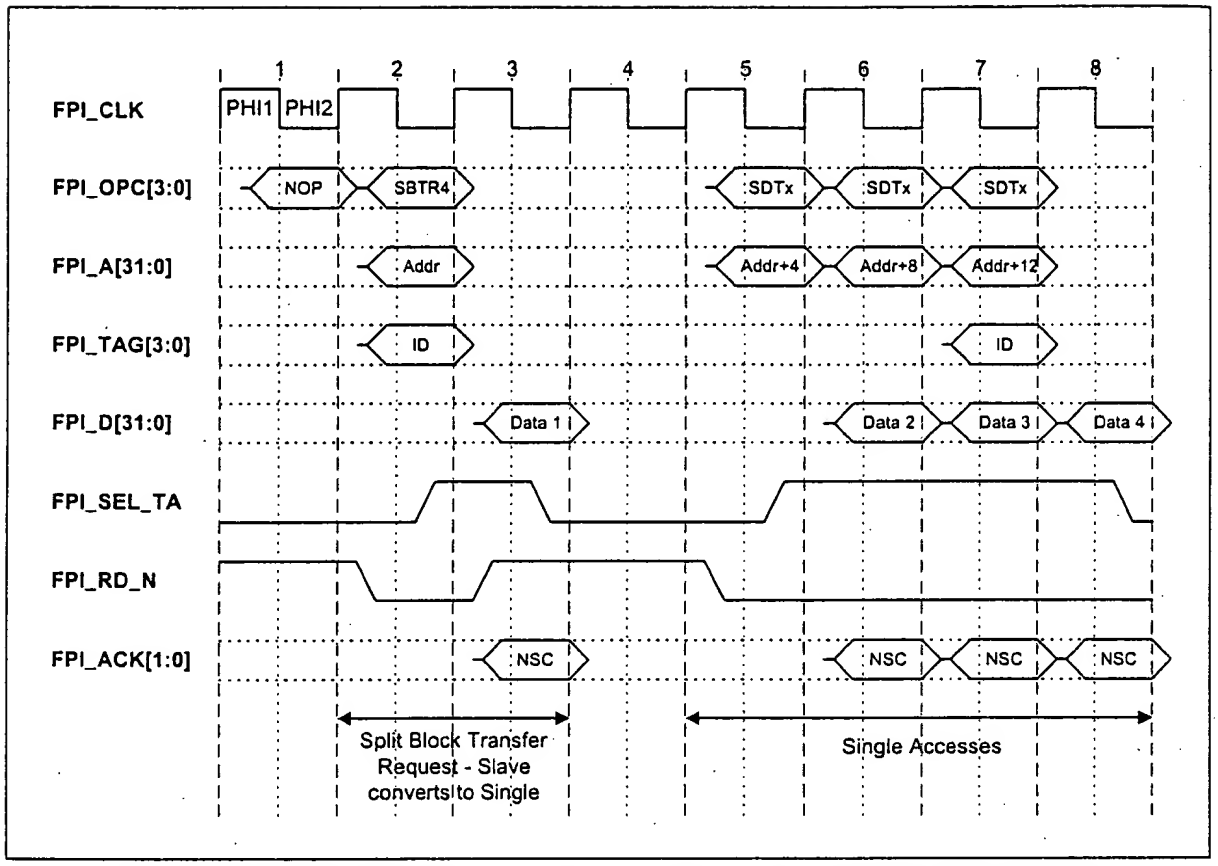

Figure 4-9: Split Block Transaction to Single Access Transformation

Cycle 2: The Split Block Transfer Request is issued by the granted master.

Cycle 3: The selected slave drives the **NSC** acknowledge code to inform the master that it does not support split block transactions. (The master shall request the bus again and read the necessary data in consecutive single accesses.) In parallel it provides the first data item.

Cycle 4: Master requests the bus again.

Cycle 5,6,7: The missing three data items are fetched with single transfers. Be aware that the address has to be incremented (figure gives example foir 32 bit implementation).

Cycle 8: Any other transaction can be started.

4.4.4 Non-split Block Read Transaction

This transaction is a kind of consecutive read operations from the same slave which must not be split by the slave.

NOTE:
It is recommended to lock Block Read Transactions to avoid any interruption.

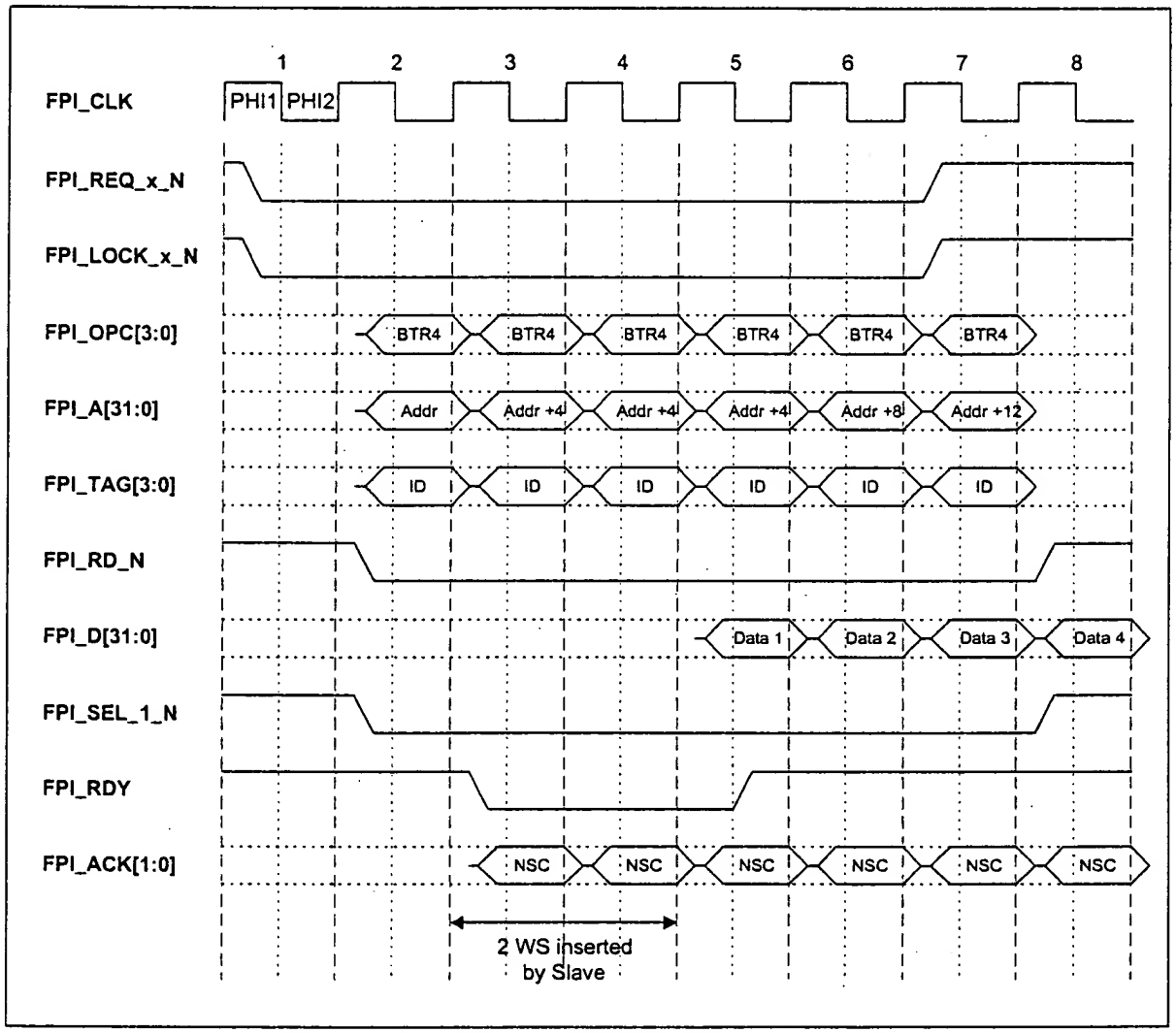

Figure 4-10: Non-split Block Read Transaction

Cycle 2:	The granted master initiates a non-split block read transaction of four data items by driving the **BTR4** code and issue the slave address.
Cycle 3, 4:	The addressed slave inserts two wait-states because it cannot provide the requested data immediately.
Cycle 5:	The transfer of data items starts with the first item.
Cycle 6, 7	The following data items are transferred. If the slave cannot provide data cycle by cycle it has to insert additional wait-states.
Cycle 8:	The last data item is transferred. In parallel a new bus operation starts.

4.4.5 Error during Block Read Transactions

In contrast to an error during single data transfers, the bus master getting an error acknowledge while performing a block transaction must not continue. It has to release the bus.
The bus controller shall issue a CPU trap to propagate the bus error to the system to invoke appropriate measures. Further action are implementation dependent.

Figure 4-11 shows this scenario:

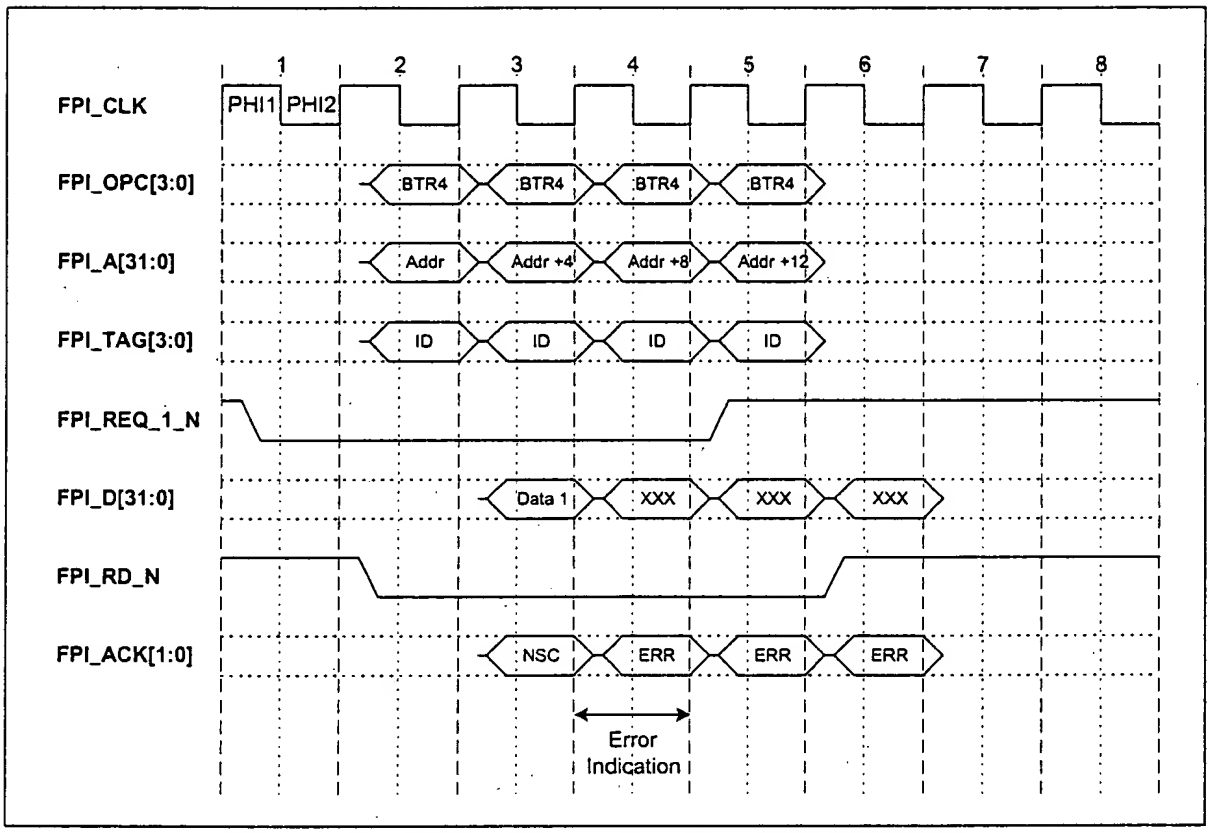

Figure 4-11: Block Transaction with ERR Termination

4.5 DMA Transfers

The purpose of a DMA controller in a system is to do transfers between system units without the overhead of interrupting the CPU with the related store and restore of registers.

Single cycle DMA transfers are possible, when there is a special (hardwired) control line to one of the related agents. In this case the DMA controller will place the address of the source and a read command onto the bus and in addition will tell the destination agent via the special control line to fetch the data from the bus during the same cycle (or vice versa).
This special control line therefore replaces the register address, the read/write control and the select line. It is not part of FPI-Bus but is part of the DMA system implementation.

In a FPI-Bus system there are two possibilities for a DMA control to use the bus for transfers:

- Single cycle transfers between any agent and some special agents (hard-wired solution as described above)
- Normal transfers (2-cycle-DMA), where the DMA controller drives the address, data and read/write lines. In the first cycle the DMA initiates a single read from the source. In the second cycle the DMA asserts the destination address and performs a single write. The DMA has to buffer the data in between these both accesses (see Figure 4-12).

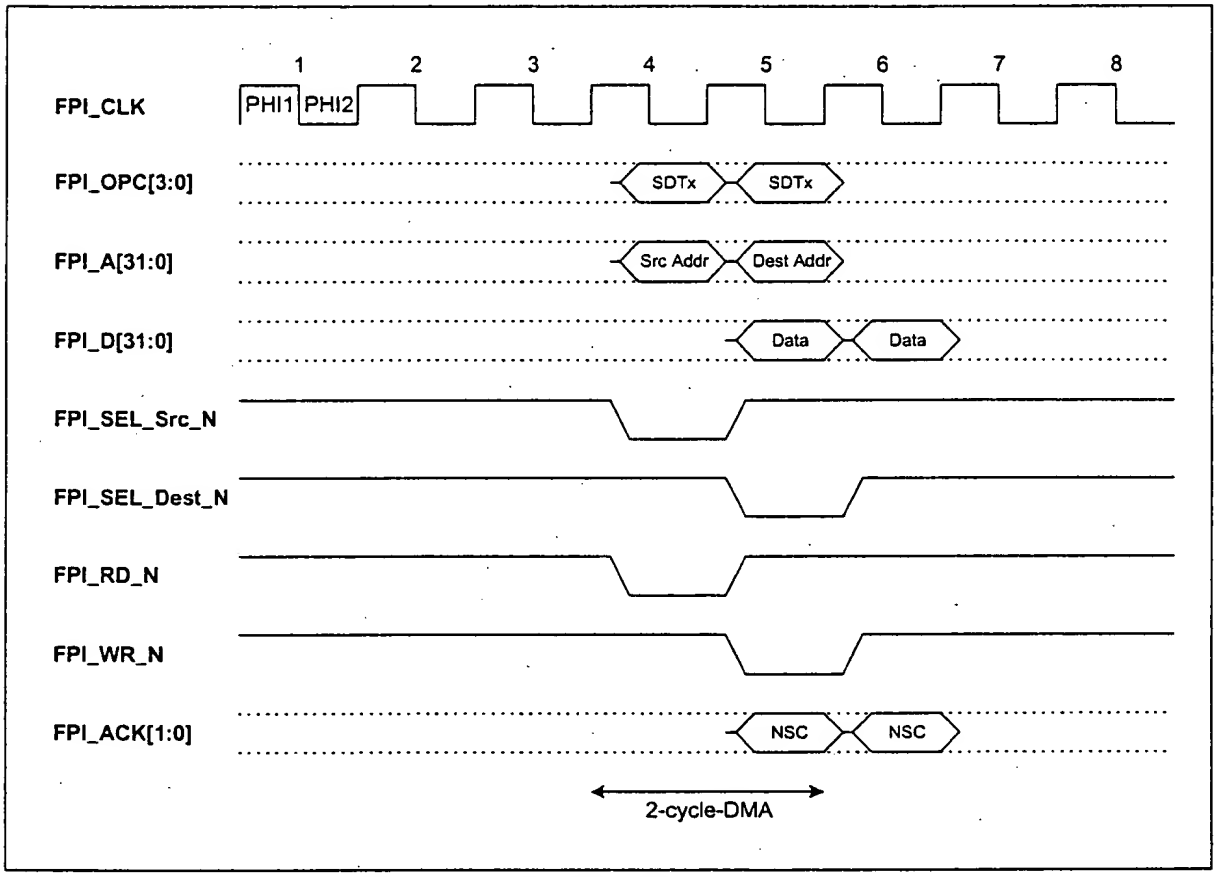

Figure 4-12: Single DMA (in 2 cycles)

NOTE:
Between the read from the source and the write to the destination the data must be buffered inside the DMA Controller.

4.6 Error Handling

Several causes for errors have been discussed. The system has to provide sufficient measures to deal with it and enable error recovery.

Each time an error condition occurred the bus controller will issue a bus error interrupt to the CPU. Debug software then has to resolve the problem or shut down the system safely.

The minimum means to detect and to handle the following errors:

4.6.1 Time-Out Error

Time-out error is detected if too many wait-states have been inserted.
If a slave drove **FPI_RDY** inactive to insert wait-states and does not drive it active again after an implementation dependent number of cycles bus control will issue **FPI_TOUT**.

In this case the selected slave has to abort the current transaction and shall release the bus in the following cycle.

In the following cycle **READY** is driven active again by Bus Control and normal bus operation goes on. For debug purposes some error information should be stored (implementation dependent).

The master which initiated the abnormal terminated transfer has to decide to repeat the transaction or do some other actions due to this abnormal transfer termination.

NOTE:
It should be avoided by implementation that this kind of transaction can be repeated again and again by master otherwise bus might be absolutely blocked and deadlock occurs.

Bus agents which are able to issue Block Read Requests shall internally detect a time-out if the requested transaction is not be answered after a implementation specific time.
For debug purposes some error information should be stored (implementation dependent).

Figure 4-13 shows the time-out handling. The following assumptions (implementation specific) are made:

- time-out detection is done centralized by bus control
- no. of max. wait-states is limited to 3 (if more than 3 wait-states are inserted time-out will be detected)
- the selected slave has started to insert wait-states one cycles ago

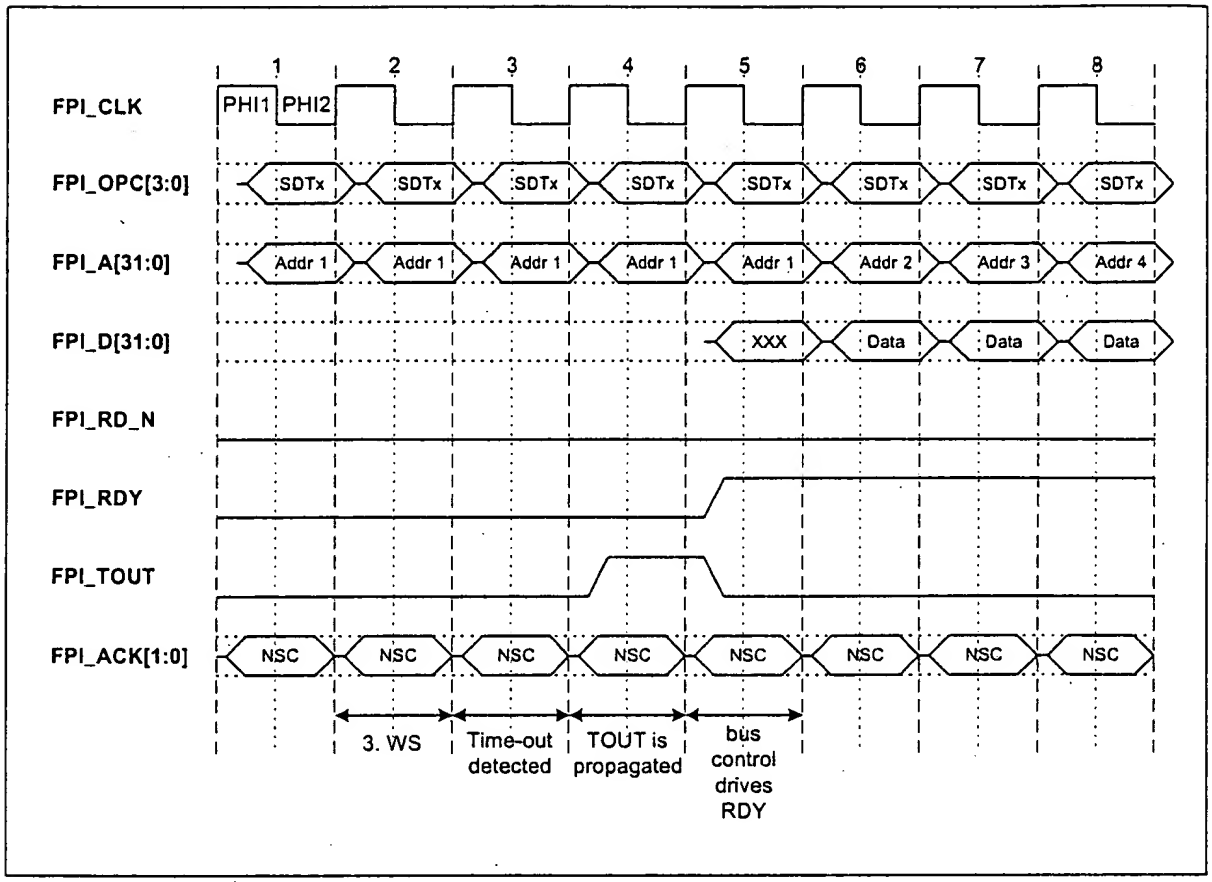

Figure 4-13: Time-out handling

Cycle 3: The 3rd wait-state is inserted still blocking the bus. Bus Control detects the time-out.

Cycle 4: **FPI_TOUT** is driven active by bus control to propagate the time-out.

Cycle 5: The bus control drives **FPI_RDY** active again and releases **FPI_TOUT**. This enables the bus for operation again.
The master that is waiting for the data must terminate the pending transfer.

Cycle 7: The normal bus operation goes on:
The data cycle of the transaction started before time-out is performed in parallel with the address cycle of the next transaction.

Cycle 8: Normal bus operation.

4.6.2 Access to non existing peripheral / to non existing register(address) in a peripheral

An access to a non existing peripheral or to a non existing register(address) in a peripheral will be **ERR** terminated. This shall cause a system trap and further actions can be taken.

The accessed device either a peripheral or the default slave (bus controller) shall not drive **FPI_RDY** inactive.

Refer to Table 3-9 for detailed information.

4.6.3 Error (ERR) Acknowledge Code

If a master receives an **ERR** acknowledge code it shall abort the current transaction.

The master has to decide to abort the following transaction whose address cycle is already in the pipe by driving **ABORT** active before the end of the address cycle or to perform it. Depending on the transaction it performs the master might get another **ERR**.

If **ERR** acknowledge code is sent a severe error occurs. This could be

- a single transaction of a defined data type which cannot be performed by the selected slave or
- any interface internal error or
- a supervisor mode access to a slave that does not support this mode.

Supervisor Mode Error

As shown in Figure 4-14 a slave that only supports supervisor mode for a given access, and is not accessed in that mode shall reply with **ERR** acknowledge code to inform the master.

The example shows an access to a user mode accessible address in cycle 2/3 and an access to an supervisor mode accessible address in cycle 3/4. The second access is **ERR** terminated because the access does not happen in the appropriate mode.

The **ERR** acknowledge leads to a bus error propagation as described above.

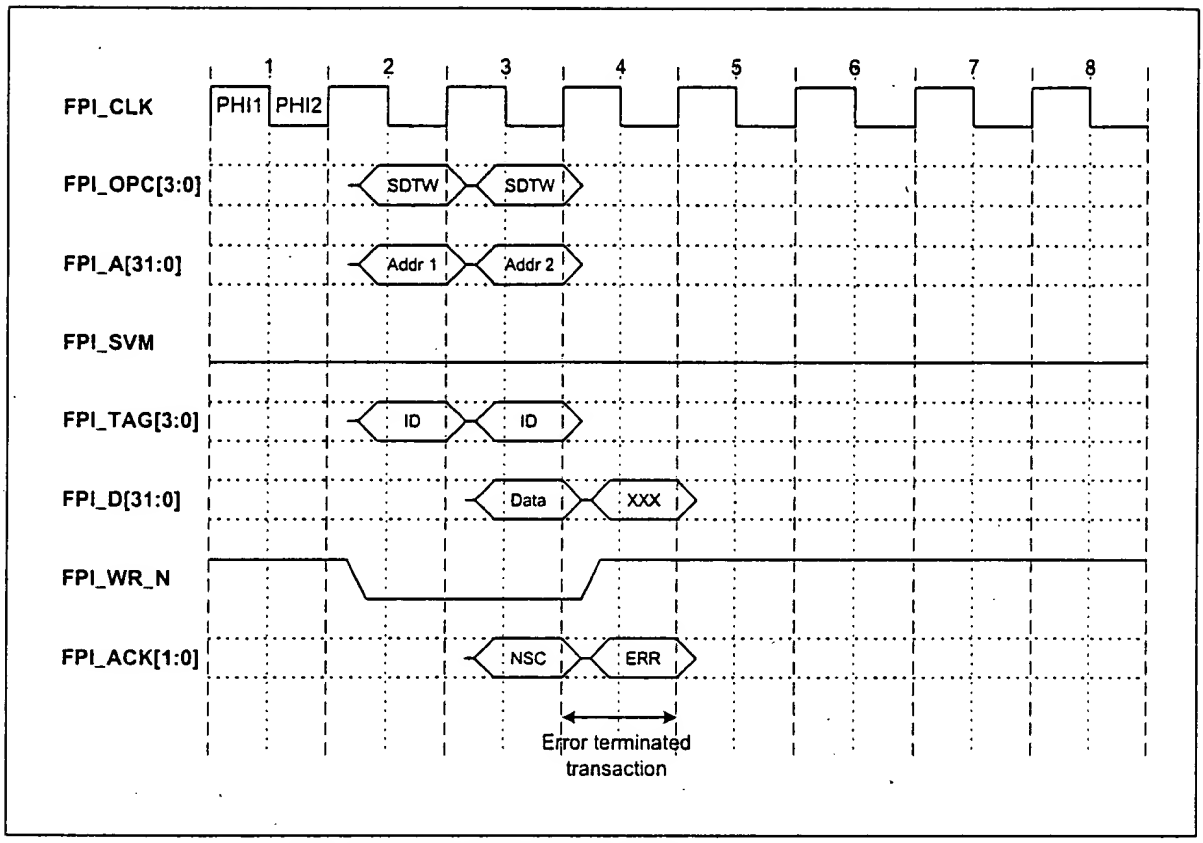

Figure 4-14: Supervisor Mode Access Error

4.6.4 Retry (RTY) Acknowledge Code

The **RTY** code informs a master to repeat the current transaction later again. The master can abort the current transaction (which address cycle has already started) by driving **ABORT** active before end of cycle.

A slave sends a **RTY** if it has already one Split Response open and is gathering data for this response when it receives a new Block Request or Single Transaction.

4.6.5 Split Block Request Failed (SBRF opcode)

If a bus agent cannot respond a Split Block Read request due to some internal errors it performs a special transfer - split block request failed (**SBRF**). This transfer is set up similar to a Split Response with length of one except of that no valid data and the opcode **SBRF** is sent.

A master which receives such a transfer shall not longer wait for a pending split response. Further actions are implementation dependent.

6.1 Deadlock Prevention Rules

6.1.1 FPI Implicit Rules

The following rules are independent of any implementation issues and are derived from the protocol.

1. A master performs only one read or one write access at a time to another device.
   *A master can have only one split response pending at any time.
   (**FPI_TAG[3:0]** bus is the limiting resource)
2. A slave can transform a read access (single or burst) to a split response.
   (not allowed in case of write, read/modify/write or non split burst)
   *Every master must be able to handle a split response.
   (minimal additional hardware effort)
3. Within a combined master/slave device the slave functionality (= another master accesses a register) must never be blocked by the master functionality (= the device itself tries to access another device), i.e. a master/slave device must still be able to accept a normal read/write access or burst request when it waits for a split response.
4. A master which requested and locked the bus must free the bus (= removing request/lock from arbiter) for at least one cycle in case of a "**SPT**", "**RTY**" or "**ERR**" acknowledge code or a time-out condition.
   (in this case the locked access sequence is broken, but a read/modify/write access will still work)

6.1.2 Rider Specific Rules

The following rules describe the implementation of the FPI interfaces on the Rider chip and might vary in the future.

1. A slave, which supports split response, will acknowledge each further access attempt with "RETRY" if it is busy with gathering data for the split response.
   *A slave serves only one master at any time.
2. The EBU has to manage the resource sharing of the external bus which can be owned by one master at any time only. A FPI access to the EBU (respectively the external devices) will be acknowledged with "RETRY", if the EBU is not owner of the external bus at that time.
   *The EBU serves only one master (internal or external) at any time.
3. Loads are always blocking because a slave might transform the read access to a split response (see 6.1.1"FPI Implicit Rules,"). Otherwise the master would have to abort an already started (address phase) second access.

6.2 Rules of Engagement

This chapter is a proposal for Rules of Engagement on the FPI Bus to guarantee Atomic and Lock Operations across different FPI platforms. It is also a compatible super-set of the existing specification

6.2.1 Goals for Atomic Operation

- Back to back, uninterrupted read followed by a write with no intervening bus operations.

6.2.2 Goals for Lock Operation

- Deterministic sequence of computation and access of peripherals and data.
- Ability to ensure completion of preceding bus events.
- Ensure other masters do not interrupt the sequence of actions in the Lock procedure.
- Avoid Deadlock

6.2.3 FPI Signals for Atomic and Lock Operation

There are several defined and proposed signals used to build Atomic and Lock Operations.

Table 6-1: Existing Atomic Signals on FPI Bus

Signal	Phase Asserted	Description
FPI_LOCK_x_N	Arbitration Round (concurrent with FPI_REQ_x_N)	A Locked Bus cycle is requested from the Arbiter. No other Masters should be granted Bus control until this Master de-asserts FPI_REQ_x_N
<FPI_RD_N, FPI_WR_N>	Address Cycle	Both signals asserted = 00 during a valid Address Cycle signify a Read-Modify-Write operation

Table 6-2: Proposed Lock Signals on FPI Bus

Signal	Source	Phase Asserted	Description
FPI_NO_SPLIT_N	Arbiter	Address phase (or more)	A Slave device may not acknowledge a read with a SPT (Split Response).
FPI_RES_REQ_x_N	Slave	Arbitration Round (in place of FPI_REQ_x_N)	Request signal for an FPI bus cycle for a Split Response. Only asserted by a Master/Slave that is able to perform a Split Response
FPI_BUSY_x_N	Slave	--	Slave capable of Split Response is currently busy completing a Split Response. Signal is de-asserted when the operation is complete and the device has returned the Split Response.

6.2.4 FPI_NO_SPLIT_N:

- The arbiter asserts FPI_NO_SPLIT_N when the bus will be, or is granted to a master that has asserted FPI_LOCK_x_N during the arbitration phase. It will remain asserted until the granted master has de-asserted the FPI_LOCK_x_N signal, even if the FPI_REQ_x_N has been periodically de-asserted.
- Slaves may not respond with a SPT response (Split) while FPI_NO_SPLIT_N is asserted, but they may complete already pending Split transactions.
- A system desiring slaves to never assert SPT may permanently tie this line asserted.

6.2.5 FPI_RES_REQ_x_N:

- Implemented *only* by Master/Slave interfaces capable of responding with SPT. (Split Response)
- Each Master / Slave capable of a Split Response uses FPI_RES_REQ_x_N to signal a bus request to the Arbiter for a Split Response bus cycle. It only uses the FPI_REQ_x_N signal to request the bus for a Master initiated bus transaction, not a Split Response.

6.2.6 FPI_BUSY_x_N:

- Implemented *only* by Master/Slave interfaces capable of responding with SPT. (Split Response)
- Each Master / Slave capable of a split response uses FPI_BUSY_x_N to signal it is busy completing a Split Response. The signal is asserted as soon as the Slave recognizes it is performing a Split Response, and is de-asserted when the pending response has completed successfully on the FPI bus.

6.2.7 FPI Arbiter Rules:

- The Arbiter may not change ownership of the bus grant to another Master once given if the FPI_LOCK_x_N was asserted during the request cycle. This is a "sticky" ownership. It may only grant to another master when the FPI_LOCK_x_N signal is de-asserted.
- The arbiter asserts FPI_NO_SPLIT_N when the bus has been granted to a master that has asserted FPI_LOCK_x_N during the arbitration phase. It will remain asserted until the granted master has de-asserted the FPI_LOCK_x_N signal, even if the FPI_REQ_x_N has been periodically de-asserted.
- When the Arbiter receives a FPI_LOCK_x_N, and that Master has won arbitration, the Arbiter asserts FPI_NO_SPLIT_N. It does not yet assert FPI_GNT_x_N to that master. It may not grant the bus to other masters. It allows any FPI_RES_REQ_x_N (Split Response) to be granted while it waits for all FPI_BUSY_x_N signals to become de-asserted. At that point it is allowed to give the FPI_GNT_x_N to the Locking master.
- The EBU may have special cases.

NOTE:
In implementations without any split transaction support the arbiter does not need to generate the FPI_NO_SPLIT_N signal.

6.3 Data Alignment

Agents have to transfer bytes and halfwords on their natural byte and halfword lanes (see **Figure 2-2 on page 22**). The byte and halfword ordering scheme is little endian.

Accesses to agents that do not support the full data bus width shall be split by master into consecutive accesses of the supported width. The bus protocol will not cover that feature.

NOTE:
In the SFR address range (peripheral accesses) word accesses on halfword boundaries are not allowed.

6.4 Bus Arbitration

Bus arbitration in a FPI-Bus based system is done by a set of request, lock and grant lines from each potential master to the arbitration logic.

6.4.1 Example for a Bus Arbiter

For arbitration each potential master(-interface) on FPI-Bus has one request and one grant line connected to bus arbiter. Masters that want to lock the bus for a certain number of consecutive transfers need an additional lock line (**FPI_LOCK_x_N**).

The request lines have different priorities that can be fixed or programmable by a set of registers inside the arbiter.

Arbitration Control does an arbitration round if any request is active and the bus is not locked and **FPI_RDY** is active.
At the end the **FPI_GNT_N** signal of the master with the highest priority of all requesting masters will be driven active.

Definition: A master is granted in cycle N if its dedicated **FPI_GNT_N** line and **FPI_RDY** was active in cycle N-1. It owns the data cycle starting in cycle N+1 ending with the cycle in which the transaction is terminated either by activating **FPI_RDY.**

If there is no request the **Default Master** will be granted.

There are several schemes possible to select a bus master for this purpose:

- the last bus master might get his grant line kept active as long there is no other request activated

- one selected implementation dependent master(e.g. CPU) can be the default master, getting the default grant
- the bus controller itself might be the implicit default master

---

NOTE:
Important is that there has to be a master on the bus for every cycle.

Figure 6-1 gives an example of an arbitration scenario.
It is assumed that the priority is decreasing from master A to master C. The default master has lowest priority.

Cycle 1: Default Master owns the bus, master A, B and C are requesting for the bus.

Cycle 2: Master A gains control over the bus because it has the highest priority and issues one opcode. Master A releases its request while B and C are still requesting.

Cycle 3: Master B gains control over the bus. Its priority is higher than the one of C. B releases its request and issues one opcode. Master C is still requesting for the bus.

Cycle 4: Master C gains control over the bus. It issues one opcode and still requests the bus to do some more transactions. At the same time master A asserts a new request.

Cycle 5: Due to the mentioned priority master A gets the bus again. It issues one opcode and releases its request. Master C is still requesting for the bus.

Cycle 6: Master C gains control over the bus again and issues another opcode. It is still requesting for the bus while there is no other request.

Cycle 7: Master C still owns the bus and issues one more opcode. It is still requesting for the bus while master B asserts a new request.

Cycle 8: Master B gains control over the bus again and issues one opcode. It is still requesting for the bus while C does the same. Because there is no request from A master B will own the bus in the following cycle, too.

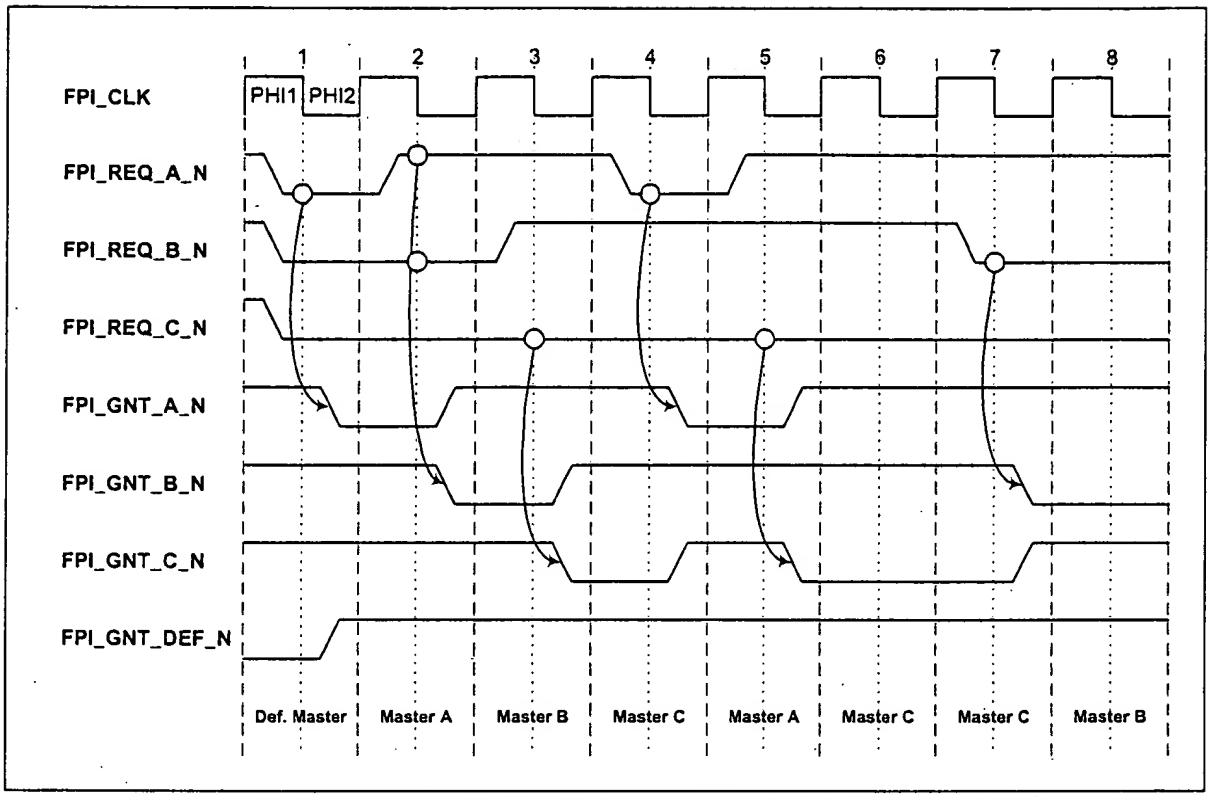

Figure 6-1: General Arbitration Scheme

NOTE:
To enable single cycle master switch bus arbiter must evaluate all request signals with the falling clock edge or combinatorial and drive a grant line before end of the cycle.
If the request signals are evaluated at the leading edge of the following cycle the master switch takes two cycles. That means the former master has to perform an idle cycle before a switch.

Implications of FPI_LOCK_N and FPI_RDY

- FPI_RDY

The arbiter shall not take care of the **RDY** signal. This has to be done by the master itself.

A granted master shall evaluate the **RDY** signal and must not start any bus actions until **RDY** is active. If **Ready** is active the granted master can start bus transactions in the following cycle.

❑ FPI_LOCK_N

A activation of **FPI_LOCK_N** will prevent a new master to gain control over the bus, even if it has a higher priority than the current bus master. The following Figure 6-2 shows the related protocol details.

Assumptions for this example:
There are three masters, a Default Master and two other masters, called A and B.
The priority of the masters is fixed: master A has the highest priority then follows B and the Default Master at the lowest.
The figure shows the case that a master activates the **FPI_LOCK_N** to prevent interruption by a higher priority master.

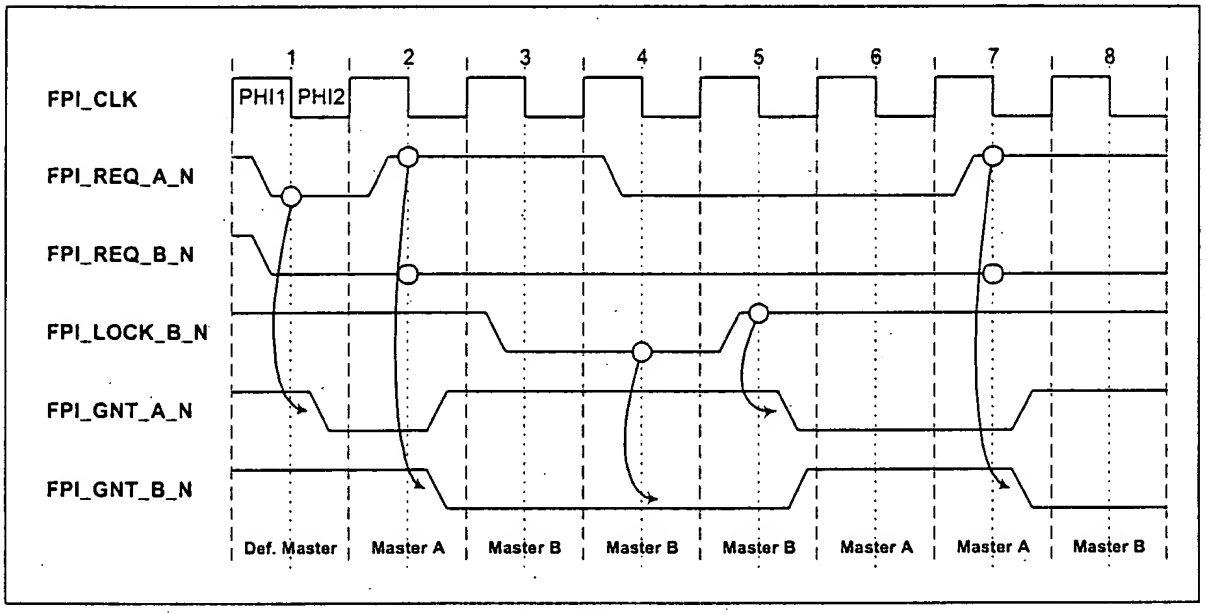

Figure 6-2: Arbitration in Case of Lock

Cycle 1: Master A and B are requesting for the bus.

Cycle 2: Master A gains control over the bus because it has a higher priority and starts one transaction. Master A releases its request while B is still requesting.

Cycle 3: Master B gains control over the bus. It starts another transaction and is still requesting for the bus. In parallel it drives its **Lock** line active.

Cycle 4: Master B is still granted. Due to the **Lock** the bus arbitration is stopped until **FPI_LOCK_N** is released again.

	It still requests for the bus while at the same time master A asserts a new request.
Cycle 5:	Although master A was requesting with higher priority master B is still granted due to the stopped arbitration. Master B releases its **Lock** line but keeps its request active. Normal bus arbitration goes on.
Cycle 6:	Master A gains control due to its higher priority.
Cycle 7:	Master A still has control over the bus. It initiates another transaction and releases it request.
Cycle 8:	Master B is granted because its still active request and the released request of master A.

6.5 Address Decoder (FPI_SEL_N generation)

Each slave interface has one dedicated **FPI_SEL_N** input.

Although the address decoding is not part of this specification it is an important issue due to the fact that only a selected slave is active on the bus.

The implemented address decoding scheme has to ensure:

- that at the end of an address cycle the appropriate **FPI_SEL_x_N** is active,
- that only one **FPI_SEL_x_N** line is active at the same time.

For security reasons it is strongly recommended to fully decode the implemented memory map. Figure 6-3 shows an example.

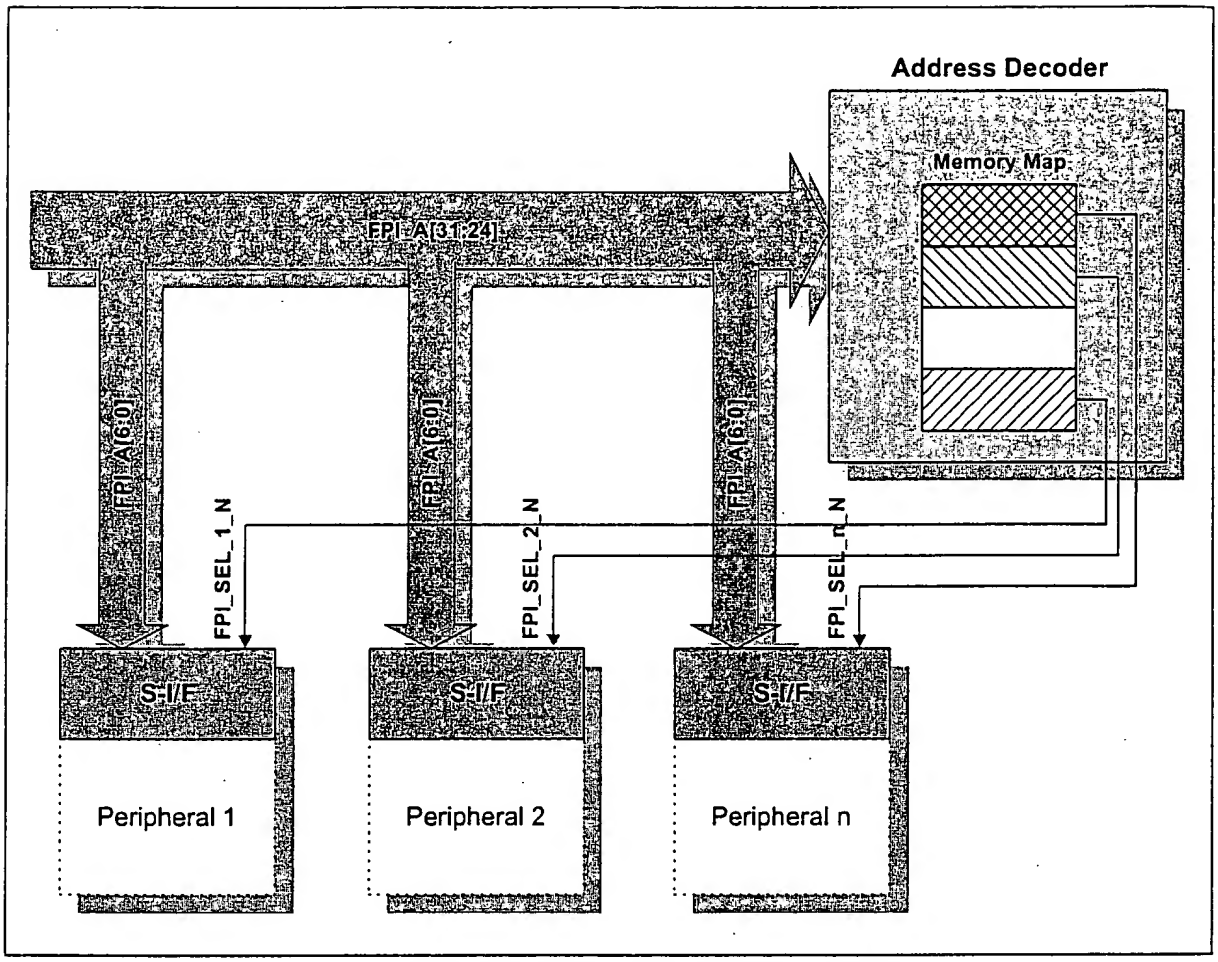

Figure 6-3: Example of Select Generation

6.6 Coupling of two FPI-Bus based Devices

To connect two FPI-Bus based devices the External Bus Controller (EBU) may be used. This unit puts the FPI access onto the external bus. The EBU of the receiving device uses its EBU to sample the access on the external bus and propagates it to the internal FPI.
The handshake of both EBUs may be done with dedicated EBU signals like chip-select, byte-enable etc.

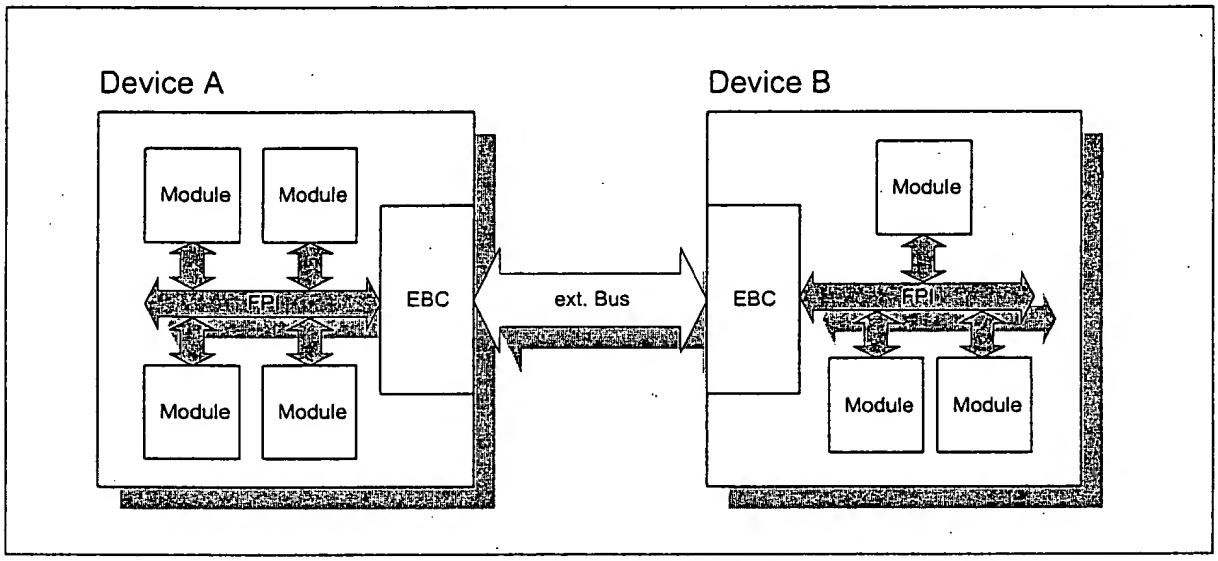

Figure 6-4: Coupling of two FPI based devices via EBU